(11)

COMPUTER NETWORK RESEARCH

ADVANCED RESEARCH PROJECTS AGENCY
SEMIANNUAL TECHNICAL REPORT

June 30, 1972

Principal Investigator: Leonard Kleinrock

Co-Principal Investigators: Gerald Estrin
Michel Melkanoff
Richard R. Muntz

Computer Science Department
School of Engineering and Applied Science
University of California, Los Angeles

114

UNCLASSIFIED

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1 ORIGINATING ACTIVITY *(Corporate author)*

Computer Science Department
School of Engineering and Applied Science 90024
405 Hilgard, University of California, Los Angeles

2a. REPORT SECURITY CLASSIFICATION

Unclassified

2b. GROUP

3 REPORT TITLE

Computer Network Research

4 DESCRIPTIVE NOTES *(Type of report and inclusive dates)*

5 AUTHOR(S) *(First name, middle initial, last name)*

Leonard Kleinrock

6 REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO OF REFS
June 30, 1972	118	180

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
b. PROJECT NO.	
c.	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*
d.	

10. DISTRIBUTION STATEMENT

Distribution of this document is unlimited.

11 SUPPLEMENTARY NOTES	12 SPONSORING MILITARY ACTIVITY

13 ABSTRACT

ARPA Semiannual Technical Report, January 1, 1972 through June 30, 1972.

DD FORM 1 NOV 68 1473 (PAGE 1)

S/N 0101-807-6801

Security Classification

Sponsored by

ADVANCED RESEARCH PROJECTS AGENCY

COMPUTER NETWORK RESEARCH

SEMIANNUAL TECHNICAL REPORT

June 30, 1972

ARPA Contract DAHC-15-69-C-0285

ARPA Order No. 1380

Principal Investigator: Leonard Kleinrock

Co-Principal Investigators: Gerald Estrin
Michel Melkanoff
Richard R. Muntz

Computer Science Department

School of Engineering and Applied Science

University of California, Los Angeles

TABLE OF CONTENTS

3

COMPUTER NETWORK RESEARCH

Advanced Research Projects Agency
Semiannual Technical Report

June 30, 1972

1. INTRODUCTION

This Semiannual Technical Report covers the period January 1 through June 30, 1972. Our activities have concentrated on computer systems studies and computer-communication network studies. ~~In~~ Section 2 ~~below, we~~ describe the results from modeling, analysis and measurement of various aspects of computer systems behavior. In Section 3 we do likewise for networks and present some specific results for the ARPA Network.

2. COMPUTER SYSTEMS STUDIES

2.1 Time-Shared Systems

Our strong effort in the area of modeling of computer systems is continuing. In the field of time-sharing algorithms, R. Muntz authored a paper entitled "Waiting Time Distribution for Round-Robing Queueing Systems" [1], and demonstrated that although round-robin has the desired effect in yielding better service to shorter jobs, it also results in vastly increased variance in response time for longer jobs, as compared with first-come-first-served (FCFS). This study investigated finite-quantum systems with overhead.

In previous reports we have discussed the importance of networks of queues as structures which can more realistically model multiple-resource systems. R. Muntz and F. Baskett (Stanford University) are currently preparing a report [2] on new results in this area. The starting point for this work was the work by Chandy on a concept called local balance. Their results include the modeling of a network in which there are different classes of customers. Customers in different classes may have different resource demand characteristics. This is useful in modeling computer systems in that different job mixes can be modeled and also in modeling computer networks in that routing of messages based on source and destination can be modeled. Their results also permit various types of scheduling disciplines at service centers in the network (viz., FCFS, RR processor sharing, LCFS). Other generalities included in these results involve state-dependent service rates and arrival rates.

This research on networks of queues is an on-going effort. Work is continuing in both extending the analytic results and in validation of the models. At least one graduate student is working on his Ph.D.

thesis in this area of research.

A Master's thesis by Walter Sheets [3], supervised by L. Kleinrock, has resulted in the generation of a simulation language and program for evaluating the performance of and experimenting with a variety of scheduling algorithms for time-shared systems. This "resource" is to become one of the standard programs available to users over the ARPA Network.

2.2 Paging Algorithms

Reference [4] (included in this report as Appendix A) considers various working set replacement algorithms. Page inter-reference interval distribution, average page fault frequency (the frequency of those instances at which an executing program requires a page of data or instructions not in the main memeory), average working set size and inter-page fault-time (time between page faults) distribution for a simulated Working Set Replacement Algorithm for three typical programs with different sizes were measured on the UCLA Sigma Executive (SEX) time-sharing system via page reference strings. These measured results are reported in this paper. The average page fault frequency relationships between working set parameters and process scheduling are discussed. These relationships are useful in planning the working set size and process scheduling which optimize system efficiency.

2.3 Buffer Behavior

A paper entitled "Buffer Behavior from Mixed Input Traffic and Single Constant Output Rate," by Chu and Liang [5], is included as Appendix B. A queueing model with limited waiting room (buffer), mixed input traffic (Poisson and compound Poisson arrivals), and constant service rate is studied. Using average burst length, traffic intensity, and input-traffic mixture rate as parameters, there are obtained relationships among buffer size, overflow probabilities, and expected message-queueing delay due to buffering. These relationships are portrayed on graphs that can be used as a guide in buffer design. Although this study arose in the design of statistical multiplexors, the queueing model developed is quite general and may be useful for other industrial applications.

2.4 Miscellaneous

Estrin, Muntz and Uzgalis published a paper "Modeling and Measurement and Computer Power" [6], which is included as Appendix C. This paper provides an informal definition of computer power and three applications of the definition to issues which will influence our ability to influence computer systems in the 1970's. For the purposes of the paper, a computer system is composed of: a centralized hardware configuration; a set of terminals for entry and exit of user programs and data; and users and user protocol for entry and exit. There is no accepted measure for global power or performance of computer systems. There is even no accepted measure for computer cost. Only when a subsystem or subfunction is isolated does it become possible to determine key parameters. However, it is useful to hypothesize such measures and consider influences on them.

In this way, the first section provides a context for the other sections by reviewing parameters which make computing systems more or less powerful. The remaining three sections of the paper are applications of the definition. The second section gives a critique of the state of modeling. The third section characterizes measurement tools. The fourth section discusses the role of measurement at the user interface.

A paper entitled "Fisheye: A Lens-Like Computer Display System," by L. Kleinrock and K. Stevens, to be published in the Communications of the ACM [7], considers the potential of a new computer display system. This system permits global vision as well as local magnification simultaneously and has shown to be quite effective in scanning large data sets.

3. COMPUTER-COMMUNICATION NETWORKS

3.1 Analysis

A paper by Frank, Kahn and Kleinrock entitled "Computer Communication Network Design -- Experience with Theory and Practice" [8] is included as Appendix D. The design of the ARPA Computer Network brought together many individuals with diverse backgrounds and philosophies. In this paper, methods used in the design of the Network are reviewed from the vantage of over two years experience in its development. The design variables, system constraints, and performance criteria for the network are discussed along with an evaluation of the tools used to design an efficient and reliable system. The design procedures and the conclusions reached about the network's properties appear to be generally applicable to message switched networks. Consequently, the results of this paper should be useful in the design and study of other store-and-forward computer communication networks.

Work by Cantor and Gerla [9] has resulted in the acceptance of their paper, "The Optimal Routing of Messages in a Computer Network via Mathematical Programming," and gives a computationally efficient exact algorithm for solving an important class of problems. The problem of finding the optimal routes for messages in a message-switched computer network can be, under proper assumptions, formulated as a nonlinear multicommodity flow problem. Many techniques that solve the most general cases can be found in the mathematical programming literature; these techniques, however, prove to be computationally inefficient for the design of a computer network. For that reason, considerable effort has been spent in the past in developing heuristic techniques. Quite satisfactory results have been obtained and the computational efficiency has been greatly improved; however, all of these techniques have various limitations. This paper presents an exact method which, by using decomposition techniques and taking advantage of the particular formulation of the problem, is computationally competitive with heuristic methods and is not affected by their limitations.

"Some Recent Advances in Computer Communications," by Chu [10], is included as Appendix E. Recent advances in computer communications are discussed, including: 1) computer traffic characteristics in the

case of short holding time representing the inquiry-response systems; 2) telephone channel error characteristics of high speed voice band data transmission on the switched telecommunication network, and of the low speed channel at a rate of 300 bits/sec; 3) optimal fixed block size for communication systems using error detection and retransmission as error control (with random or burst error channel); 4) statistical multiplexing (Asynchronous Time Division Multiplexing); 5) loop systems; and 6) security in computer communications. New areas needing further investigation are included.

G. Fultz completed his work on "Adaptive Routing Techniques for Message Switching Computer-Communication Networks" [11] under the supervision of L. Kleinrock. This Ph.D. thesis will be published as one of our Computer Systems Modeling and Analysis Group reports and will enjoy wide distribution. This report considers adaptive routing techniques applicable to message switching computer-communication networks such as the ARPA Network. The emphasis is on the prediction of average message delay and the specification, implementation and evaluation of various classes of message routing procedures. The fundamental operational aspects of a message switching network model are presented. This model, which is based upon the existing ARPA Network, is utilized for the formulation of theoretical network performance measures and is the basis for a computer simulation program written to obtain the performance of specific routing algorithms. A methodology for the investigation of message routing strategies applicable to message switching networks is developed and six key areas requiring study are identified. Known routing techniques are classified into three broad catagories: 1) deterministic routing techniques, 2) stochastic routing techniques, and 3) flow control routing techniques. From this classification, one can determine which techniques are applicable to theoretical studies and which are candidates for operating network algorithms. The remainder of the report is concerned with the following investigations: 1) network performance measures and models, 2) specification, implementation and evaluation of deterministic, stochastic and flow control routing algorithms, and 3) the impact of a network's size and topology on message routing procedures.

3.2 Measurement Activities

During the period of March 1 through June 30, 1972, steps were taken to accelerate and broaden the Network Measurement effort, mainly through the effort of V. Cerf. Two major goals are:

a) the creation of a network measurement laboratory
b) execution of as wide a variety of measurement experiments as possible

These goals are not restricted to UCLA's Network Measurement Center, but also include other interested sites around the network. In order to coordinate network-wide measurement efforts, a Network Measurement Group was formed in mid-March. As chairman of this group, V. Cerf has sought to initiate cooperative measurement experiments among the network sites. Two early projects arising from the formation of this group include the

installation of NCP measurement software at several sites (UCLA, DMCG, Lincoln Labs), and the provision of HOST message discard mechanisms. Artificial HOST traffic generators are to be installed during the summer at cooperating sites (UCLA, SDC, Lincoln Labs, DMCG). Another project which is very nearly operational is for DMCG to collect HOST availability statistics (automatically, as opposed to the manual method in use at BBN). UCLA also has such a survey program in operation. DMCG will collect results from its program and from ours at NMC, and make the results available (perhaps on-line) in network measurement notes.

Experiments at UCLA have centered on two aspects of the network during this period. The first is an investigation on network behavior under heavy local load. The experiment goes by the name RAND SATURATION EXPERIMENT and was initiated by W. Naylor at UCLA. The results have been published as Network Measurement Note #2 [12] and are included as Appendix F. Further saturation experiments will be performed under the new IMPSYS as soon as it is installed.

A second experiment, the TINKER-McCLELLAN experiment, has involved monitoring of actual traffic between TINKER AFB and McCLELLAN AFB. Data are automatically gathered and analyzed under program control after initiation of the experiment. A preliminary report is in preparation which will compare results obtained by NMC and results published by TINKER and McCLELLAN. As with the saturation experiment, these transmissions will be monitored regularly, and results for the New IMPSYS (62600) will be compared with the old (2510).

Theoretical results obtained by Kleinrock, Cole and Fultz on the distribution of inter-packet time gaps at the receiving IMP will be tested by actual experimentation during this period; comparative results will be available after installation of the new IMPSYS.

With respect to the design and implementation of a measurement laboratory, the following software supports have been or are in the process of being constructed.

1) NETSTAT: A program which collects statistics messages from the NCP and formats them into a standardized file format.

2) NETSET: A program which permits experimenters to manually initiate and terminate measurement at the IMP's in the network. Artificial traffic generators can also be started and stopped at various IMP's through the use of this program. This is a fundamental measurement tool and plays an important role in most of our experiments.

3) SATIMP: This program accepts as input a simple file which describes the entire course of a RAND saturation experiment. Once SATIMP is started, data is collected automatically and analyzed, message generators are started and stopped, all under program control.

4) NWC and NWMP: These programs permit a user to sequence through the data obtained by NETSTAT, accumulating data and printing out selected statistics messages (or trace or snapshot messages) obtained from IMP's. This facility is essentially a low level filtering and formatting service.

5) TMPROC: This program accepts as input files from NETSTAT and produces statistical analyses of traffic length distribution, delays and transmission rates, and link utilization observed during transmissions between TINKER AFB and McCLELLAN AFB. The results are published separately for traffic from TINKER to McCLELLAN and vice-versa.

6) SURVEY: This program automatically polls the HOSTS in the net and collects status information, delay to perform ICP (if ICP is possible). The resulting data are placed in a file for further processing (BY DMCG, eventually).

7) GRAPHIT: This program utilizes the IMLAC Fortran graphics package and permits us to present our results graphically on the two available IMLAC terminals. Eventually this facility will permit interactive monitoring of experiments while they are being run. The facility will be helpful at the ICCC in October.

The network measurement effort has benefited from the strong support of the SPADE software staff. In particular, the installation of a time-of-day clock, a high priority "super queue," and a mechanism for scheduling interrupts at pre-determined times have made it possible to design a HOST traffic generator for the SEX system. A programming philosophy is taking shape, partly as a result of J. Postel's and M. Kampe's programming work. This philosophy advocates the use of 'HELP' routines embedded in all user level programs which will respond to the bewildered user's cry for help with increasingly detailed instructions and explanations. The mechanism to provide this service will be common to all programs and will permit programmers to create and modify their HELP sections through the use of the EDIT program, without the need to recompile or reassemble any working program. The ease with which such help sections can be provided will contribute to the quality and quantity of these facilities.

ONGOING PROJECTS

1) Specification of an artificial traffic generator for SDC's DDP-516.

2) Installation of the SMOG package in the SEX system (under the direction of C. Maxwell).

3) Design of the Message Switching Protocol Instrumentation (with D. Walden of BBN).

4) Preparation of papers for the following conferences:

a) COMPCON 72: "Selected ARPANET Measurement Experiments" with W. Naylor.

b) WESCON 72: "Selected ARPA Network Measurements" with W. Naylor.

These papers will contain the results of different experiments performed on the net.

5) Preparation of a measurement demonstration for ICCC (in conjunction with R. Kahn, BBN).

3.3 Software Development

This period has been one of a high level of involvement in ARPA Network protocol development. J. Postel has been particularly active in Telnet and Remote Job Entry protocol issues and has also participated in File Transfer and Graphics protocol development. He co-authored a paper on "Function Oriented Protocols for the ARPA Computer Network" [13] which is attached as Appendix G. Furthermore, the UCLA-NMC implementation of the user-Telnet program has been augmented by adding facilities to send and receive data from/to files. This is a very flexible arrangement and is further explained in a section of the SEX Notebook (the system reference manual) titled "The Telnet Switch."

4. CONCLUSIONS

This period, then, has seen some exciting advances in the modeling of computer systems, particularly in the creation of the general model for queueing networks. The ARPA Network measurement effort has accelerated considerably and numerous measurements have been reported already; ongoing experiments continue at present. The analytical progress in networks is also moving along rapidly and new results are forthcoming. The symbiosis between modeling, analysis and measurement continues to be a healthy one, and we offer our progress as a prime example of the mutual benefit one achieves in such a relationship.

PUBLICATIONS

1. Muntz, R., "Waiting Time Distribution for Round-Robin Queueing Systems," PIB International Symposium XXI on Computer-Communications Network and Teletraffic, April 1972.

2. Muntz, R., and F. Baskett, "Open, Closed, and Mixed Networks of Queues of Different Classes of Customers," to be published.

3. Sheets, W., "A Simulator for a Large Class of Scheduling Algorithms for Time-Shared Computers," Master's thesis, Computer Science Department, School of Engineering and Applied Science, University of California, Los Angeles, September 1972.

4. Chu, W. W., N. Oliver, and H. Opderbeck, "Measurement Data on the Working Set Replacement Algorithm and Their Applications," PIB International Symposium XXI on Computer-Communications Network and Teletraffic, April 1972.

5. Chu, W. W., and L. C. Liang, "Buffer Behavior for Mixed Input Traffic and Single Constant Output Rate," IEEE Trans. on Communications, Com-20, No. 2, April 1972, pp. 230-235.

6. Estrin, G., R. R. Muntz, and R. C. Uzgalis, "Modeling and Measurement and Computer Power," AFIPS Conference Proc., Spring Joint Computer Conference 1972, 40:725-738.

7. Kleinrock, L., and K. Stevens, "Fisheye: A Lens-Like Computer Display System," accepted for publication in Communications of the Association for Computing Machinery, 1972.

8. Frank, H., R. E. Kahn, and L. Kleinrock, "Computer Communication Network Design -- Experience with Theory and Practice," AFIPS Conference Proc., Spring Joint Computer Conference 1972, 40:255-270.

9. Cantor, D., and M. Gerla, "The Optimal Routing of Messages in a Computer Network Via Mathematical Programming," IEEE Computer Science Conference 72, San Francisco, California, September 12-14, 1972.

10. Chu, W. W., "Some Recent Advances in Computer-Communications," to be presented at USA-Japan Computer Conference, Tokyo, October 3-5, 1972.

11. Fultz, G., "Adaptive Routing Techniques for Message Switching Computer-Communication Networks," Ph.D. dissertation, Computer Science Department, School of Engineering and Applied Science, University of California, Los Angeles, June 1972.

12. Cerf, V., and W. Naylor. "Rand Saturation Experiment Preliminary Results," Network Measurement Note #2, Computer Science Department, University of California, Los Angeles.

13. Crocker, S., J. F. Heafner, R. M. Metcalfe, and J. Postel, "Function-Oriented Protocols for the ARPA Computer Network," AFIPS Conference Proc., Spring Joint Computer Conference 1972, 40:271-279.

Additional Publications and Presentations

Kleinrock, L., "Computer Networks," in Computer Science, A. F. Cardenas, L. Presser, and M. A. Marin (eds.), Wiley Interscience, New York, 1972, pp. 241-284.

Kleinrock, L., "Survey of Analytical Methods in Queueing Networks," in Computer Networks, R. Rustin (ed.), Courant Institute Computer Science Symposium III, Prentice-Hall, Englewood Cliffs, New Jersey, 1971.

Kleinrock, L., "Performance Models and Measurements of the ARPA Computer Network," presented at "Reseaux de Calculateurs," Paris, France, May 25-26, 1972.

Cerf, V., "Network Measurement," presented at Conference on Resource Sharing Computer Networks, Montreal, May 31, 1972.

Master's Theses

Kline, C., "LISP Interpreter in a Paged Environment," Master's thesis, Computer Science Department, School of Engineering and Applied Science, University of California, Los Angeles, Dec. 1971. Chairman: R. R. Muntz

Sei, K., "Software Design for the Lincoln Wand," Master's thesis, Computer Science Department, School of Engineering and Applied Science, University of California, Los Angeles, March 1972. Chairman: L. Kleinrock.

Sheets, W., "A Simulator for a Large Class of Scheduling Algorithms for Time-Shared Computers," Master's thesis, Computer Science Department, School of Engineering and Applied Science, University of California, Los Angeles, Sept. 1972. Chairman: L. Kleinrock.

Opderbeck, H., "On Program Behavior and the Efficiency of Replacement Algorithms," Master's thesis, Computer Science Department, University of California, Los Angeles, Dec. 1971. Chairman: W. W. Chu.

Tobagi, F., "Comparison of Various Queueing Network Models," Master's thesis, Computer Science Department, University of California, Los Angeles, Dec. 1971. Chairman: L. Kleinrock.

Wong, J., "Performance Evaluation of the SEX Time-Sharing System," Master's thesis, Computer Science Department, School of Engineering and Applied Science, University of California, Los Angeles, Dec. 1971. Chairman: R. R. Muntz.

Ph.D. Dissertations

Cerf, V., "Multiprocessors, Semaphores, and a Graph Model of Computation," Ph.D. dissertation (Report No. UCLA-ENG-7223), Computer Science Department, School of Engineering and Applied Science, University of California, Los Angeles, June 1972. Chairman: G. Estrin.

Dobieski, W., "Closed Cyclic Queues," Ph.D. dissertation, Computer Science Department, School of Engineering and Applied Science, University of California, Los Angeles, June 1972. Chairman: L. Kleinrock.

Fultz, G., "Adaptive Routing Techniques for Message Switching Computer-Communication Networks," Ph.D. dissertation, Computer Science Department, School of Engineering and Applied Science, University of California, Los Angeles, June 1972. Chairman: L. Kleinrock.

APPENDIX A

MEASUREMENT DATA ON THE WORKING SET REPLACEMENT ALGORITHM AND THEIR APPLICATIONS

by W. W. Chu, N. Oliver and H. Opderbeck

Presented at the PIB International Symposium XXI on Computer-Communications Network and Teletraffic, April 1972.

Measurement Data on the Working Set Replacement Algorithm and Their Applications*

by

W.W. Chu, N. Oliver† and H. Opderbeck

Computer Science Department
University of California
Los Angeles, California 90024

ABSTRACT

Page inter-reference interval distribution, average page fault frequency (the frequency of those instances at which an executing program requires a page of data or instructions not in the main memory) average working set size and inter-page fault-time (time between page fault) distribution for a simulated Working Set Replacement Algorithm for three typical programs with different sizes were measured on the UCLA Executive (SEX) time-sharing system via page reference strings. These measured results are reported in this paper. The average page fault frequency relationships between working set parameters and process scheduling are discussed. These relationships are useful in planning the working set size and process scheduling which optimize system efficiency.

*This research was supported by the U.S. Office of Naval Research, Mathematical and Information Sciences Division, Contract No. N00014-69-A-0200-4027, NR 048-129 and the Advanced Research Projects Agency of the Department of Defense, Contract No. DAHC 15-69-C-0285

†Formerly of UCLA, now at General Motors Research Technical Center, Warren, Michigan.

I. Introduction

Memory management becomes a severe problem in multiprogramming and virtual memory systems. In a multiprogramming system, many programs are concurrently executed by the processor. Thus the main memory is shared by many programs. Since the total size of all of the programs far exceeds the size of the main memory, in order to keep information that will be used in the near future in the main memory, the system constantly moves information between several levels of storage media.

In this paper, we consider the case of paged memory systems: that is, the address spaces are partitioned into equal size blocks of contiguous addresses. The paged memory system has been used by many computer systems. However, the basic page replacement problem of deciding which page should be kept in main memory and which should be removed when additional space is needed is still little understood and has been of considerable interest. Obviously, the page removed should be a page with the least probability of being needed in the near future. The difficulty lies in trying to determine which page this will be without incurring difficult implementation problems at the same time.

Many replacement algorithms have been proposed and studied in the past: such as Random, First-in First-out, Stack Replacement Algorithms[1] (for example, Least Recently Used (LRU)), and the Working Set Replacement Algorithm.[2] The first three replacement algorithms require a fixed size memory space for each process. The Working Set Replacement Algorithm, however, requires a variable size storage space for each process and the size

varies with program demands. This variable storage space provides an adaptive capability in the replacement algorithm which is quite appealing. The working set principle of memory management states that a program may use a processor only if its working set (set of pages) is in the main memory, and no working set pages of an active program may be considered for removal from the main memory. Properties of the working set replacement algorithm, the relationships among page inter-reference interval, average page fault frequency and average working set size for the Working Set Replacement Algorithm are described in a recent paper by Denning and Schwartz.[3]

Because of the complex nature of program behavior, analytical estimation of the above mentioned parameters of program behavior becomes very difficult. Yet this information is important in the planning of an efficient replacement algorithm that optimize system performance. Therefore we employ measurement techniques for such estimations. We collect data about the pattern of references to all the pages which comprise the executed program, and measure these parameters experimentally via interpretive execution. This technique has been used previously to measure dynamic program behavior[4] and also to measure the performance of Belady's Optimal Replacement Algorithm[5] and LRU replacement algorithms.[6,7]

Here we report the measured program behavior of the Working Set Replacement Algorithm. We shall first report measurement results such as page inter-reference interval distribution, average page fault frequency, average working set size and inter-page-fault-time distribution. We then discuss the use of average page fault frequency to determine the working set parameter, and propose a page fault scheduling algorithm for process scheduling which improves system efficiency.

II. Measurements and Results

The working set $W(t,\tau)$ at a given time t is the set of distinct pages referenced in the time interval $((t-\tau+1), t)$, where τ is called the working set parameter. The working set size $w(t,\tau)$ is the number of pages in $W(t,\tau)$. The average working set size $S(\tau)$ defines as $S(\tau) = \lim_{k\to\infty} \left\{ \frac{1}{k} \sum_{t=1}^{k} w(t,\tau) \right\}$. For systems employing working set replacement algorithms, several parameters of interest are: 1) page inter-reference interval distribution $F(\tau)$ which describes the fraction of the page inter-reference intervals less than τ; 2) average page fault frequency $m(\tau)$ which describes the average number of page faults per page reference for working set parameter τ; 3) average working set size $S(\tau)$; and 4) inter-page-fault-time (time between page fault) distribution $P(t,\tau)$ which describes the fraction of the inter-page-fault-times less than or equal to t for a given τ.

$F(\tau)$ is a fundamental distribution; it closely relates to the other three parameters. When we assume that the page reference rate is one page per unit time, we know that the page references that result in page faults are those references whose inter-reference intervals exceed τ. Thus, $m(\tau) = 1-F(\tau)$. It can be shown[3] that $S(\tau) = \sum_{z=0}^{\tau-1} m(z)$. Thus, $S(\tau)$ is closely related to $m(\tau)$. $1/m(\tau)$ is the average running time between page faults. Since $P(t,\tau)$ is the fraction of inter-page-fault-time less than or equal to t, $1/m(\tau)$ is the time average of the density function $P(t+1,\tau) - P(t,\tau)$; that is, $1/m(\tau) = \sum_{t=1}^{\infty} t\cdot[P(t+1,\tau) - P(t,\tau)]$.

To employ measurement techniques for estimating these parameters, we collect data about the pattern of references to all the pages which comprise the executed program and measure these parameters experimentally via interpretive execution. For this purpose an interpreter for the UCLA Sigma-7 time-sharing system has been developed. This interpreter is capable of

executing Sigma-7 object programs by handling the latter as data and reproducing a program's sequence of references. This sequence, in turn, can then be used as input to programs which simulate the Working Set Replacement Algorithm.

Three different programs with different sizes were interpretively executed, and their behavior was investigated under the Working Set Replacement Algorithm. A FORTRAN Compiler was chosen as the representative for a small program. META-7 was chosen as the representative for a large program. It translates programs written in META-7 to the assembly language of the Sigma-7. A DCDL (Digital Control Design Language) compiler was chosen as a representative for a medium size program. This compiler is written in META-7. DCDL translates specifications of digital hardware and microprogram control sequences into interpretive code.

Table 1 shows some characteristic properties of these programs. The column 'size' is divided into two parts. 'Static' refers to the number of pages necessary to store the program as an executable file on a disk where one page consists of 512 32-bit words. 'Dynamic' indicates the number of different pages actually referenced while processing the given input data. The difference between the number of pages in static and dynamic results from the fact that programs creat new pages during execution for working storage areas and that not all pages of programs are reference during executing a specific set of input data.

Table 1. Program sizes of the three measured programs

	Size		Number of page references
	Static	Dynamic	
FORTRAN	24	34	1,000,000
DCDL	44	58	1,000,000
META-7	84	153	1,000,000

Figure 1 shows the average page fault frequency $m(\tau)$ for the three programs. We note that all three programs exhibit similar page fault characteristics. The average page fault frequency decreases rapidly with τ. Large programs tend to have a slower rate of decrease. The reason for such characteristics is mainly the locality of the program; that is, during any interval of execution, a program favors a subset of its pages, and this set of favored pages changes its membership slowly. Further, the locality for large programs is usually larger than that of small programs. The page inter-reference interval distribution $F(\tau) = 1-m(\tau)$ can be obtained easily from $m(\tau)$. The average working set sizes as a function of τ are shown in Figure 2. Measurement data support the premise that average working set size increases as program size increases and reaches a constant level as τ reaches a certain value. The $P(t,\tau)$'s of the three programs for selected τ's are shown in Figure 3. We note that $P(t,\tau)$ is very sensitive to τ and program size. For a given program, the average inter-page-fault-time increases as τ increases. This occurs because for the small τ case, many of the pages to be referenced in the near future are in the secondary memory; thus the average working set size is very small and yields a high page fault rate. For the large τ case, most of the pages are in the main memory which yields a large average working set size and a small page fault rate. For

a given τ, large size programs have a higher page fault rate than that of a small size program. In the next section we shall discuss the applications of these parameters to determine the working set parameters and process scheduling which improve system efficiency.

III. Applications of Measurement Data

A. Working Set Parameter τ

Working Set Parameter τ is an important parameter which affects page fault rate, memory utilization, and thus system efficienty. The measurement data support the fact that τ should be chosen according to the executing program (e.g., locality) and system organization (e.g., available memory size and the speed ratio between main and secondary memory). If τ is not properly chosen, for example if τ is too short, then pages are removed from the main memory while still potentially useful. This results in high page traffic between the different levels of memory. If τ is too long, then pages that are not needed may remain in the main memory, which is an inefficient use of memory space. Instead of choosing τ arbitrarily, we propose to determine τ from the measured $m(\tau)$ and designate it as τ^0. As a result, τ^0 is now closely related to program behavior as well as to system organization.

The efficiency of a program is defined as the ratio of total virtual running time to total real running time (total virtual time and total page waiting time); that is,

$$\text{Eff} = \frac{\text{total virtual running time}}{\text{total real running time}} \tag{1}$$

$$= \frac{1}{1+m(\tau)R}$$

where $R = A/T$

A = Access time of the main memory

T = Access time of the secondary memory

Since R is fixed for a given system, from (1) we know a fixed average page fault frequency $m(\tau)$ insures a certain level of efficiency.

Suppose we would like the system to operate at an average page fault level of about 10^{-4} page faults/reference; that is, one page fault in every 10^4 page references. Then from Figure 1, τ^0 for Fortcomp, DCDL and META-7 are 22, 45, and 54 m sec (1 μsec per page reference) respectively. From Figure 2, the corresponding average working set size is 15, 36, and 39 pages.

Usually in a multiprogramming environment several types of programs may be concurrently operated by the operating system. The working set parameter of such a system may either be variable of fixed. In the variable τ case, the τ^0 should change from one program to another; while in the fixed τ case, the τ^0 remains fixed for all types of programs. Because of the simplicity of a fixed τ scheme, it requires less overhead to implement than the variable τ scheme. However, the efficiency may not be as high as that of the variable τ case.

One way to determine the value of a fixed τ is to use the weighted average working set parameters of each program; that is,

$$\tau^{o} = \frac{1}{n} \sum_{i=1}^{n} u_i \tau_i^{o} \qquad (2)$$

where τ_i^{o} = working set parameter for the i^{th} program that selected from its $m(\tau)$

u_i = relative usage frequency of the i^{th} program

n = total number of distinct programs used in the system

The decision as to which scheme should be used for a given system should be based on program behavior, relative usage frequency of all the distinct programs used by the system, and the overhead in implementing these schemes.

B. Process Scheduling

In a multiprogramming system, to increase system efficiency and to reduce response time for short jobs, the job queues for CPU processing usually have several priority levels. Let us consider a system having two levels of queues: Short Quantum Queue (SQQ) and Long Quantum Queue (LQQ). SQQ has a higher priority than LQQ. All jobs enter the SQQ. Processes in the SQQ are given one time slice at a time. The process is put at the back of the SQQ after the process either incurred a page fault or used up the time slice; that is, the process is serviced in a round-robin fashion. A process stays in the SQQ until its short quantum time runs out. It is then put on the front of the LQQ. The LQQ will not be serviced until the SQQ is empty. A process in the LQQ receives service until its long quantum time runs out. It is then put at the end of the LQQ.

When a system is properly designed, such scheduling algorithms yield: 1) fast response time to short jobs, and 2) most of the short jobs are run in the SQQ and long jobs (compute-bound processes) will run in the

LQQ. Since LQQ provides more memory space for each process than that of SQQ, such scheduling yields less page swapping.

If the quantum time of the SQQ is too short, then many of the short jobs will be in the LQQ; if the quantum time is too long, then many computational jobs will be in the SQQ. The system is designed such that most of the short jobs finish their processing in the SQQ and only the compute-bound processes enter into the LQQ. The short quantum time should be larger than the average real process time of short jobs. However, the process time varies from one process to another. In addition, the processing time is further complicated by page faults occurring during its execution.

The real processing time of a process is the sum of the virtual process time and the total time wasted* due to page faults of that process. For example, two processes requiring the same amount of virtual CPU processing time could have very different page fault frequences, and thus yield very different real processing time. Therefore the real processing time is extremely difficult to estimate.

We know that page fault frequency has great influence on system efficiency and the response time of the short jobs. We propose to use a page fault as a measure in process scheduling; that is, when a process exceeds a certain number of page faults or exceeds the quantum time of the SQQ (whichever occurs first), then the process switches from the SQQ to the LQQ. We shall call such a scheme a page fault scheduling algorithm. In a multiprogramming environment, the CPU idle times due to page swapping between main and secondary memories are directly affected by the page fault frequency. The page fault scheduling algorithm should be effective in reducing CPU idle time And improve system efficiency. (See Appendix).

---

*For a system operating in a multiprogramming environment, we should also include the time spent in waiting for the availability of CPU.

Processes with high page fault rates occupied in the main memory greatly reduce the efficient utilization of main memory. The page fault scheduling algorithm adaptively allocates the low page fault rate processes in the main memory and higher page fault rate processes in the secondary memory. Thus such scheduling improves the utilization of main memory. As a result, this will improve the average response time of the system. An analogy to the above scheduling algorithm is the well known "serving the shortest job first" algorithm in queueing theory that results in improvements in average waiting time; except in our case we have further improved the memory utilization efficiency.

The number of page faults occurring during processing before switching a process from a SQQ to a LQQ depends on the response time required, the number of processes operating concurrently, the replacement algorithm used, and page fault frequency characteristics. Further study in this area is needed.

In order to reduce response time, the quantum time of the SQQ and LQQ are further divided into many time slices. The optimal size of time slices is another important parameter that affects system efficiency. The time slice should be selected such that most of the processes either page fault or become inactive before running out of the time slice. Since $P(t,\tau)$ describes the inter-page-fault-time distribution of a process for a given τ, the time slice for the Quantum Queues can be determined from $P(t,\tau)$. For example, if we wish 95% of the time that the process will page fault before running out of the time slice -- that is, only 5% of the time the process will run to the end of the time slice -- then from Figure 3 we know

the time slices of the LQQ* for τ = 10 m sec are: 28 m sec for the FORTRAN Compiler, 13 m sec for DCDL, and 12 m sec for META-7. Time slices for τ = 25 m sec are: 58 m sec for the FORTRAN Compiler, 38 m sec for DCDL, and 35 m sec for META-7. Thus, the measured inter-page-fault-time distribution provides a good way to determine the optimal time slices for the Quantum Queues which avoids excessive unnatural interrupts that degrade response times.

The page fault scheduling algorithm, as well as the selection of the time slice form inter-page-fault-time distribution, are quite general and can be applied to other types of replacement algorithms.

V. Conclusions

Page inter-reference interval distribution, average working set size, average page fault frequency, and inter-page-fault-time distribution for three typical programs with working set replacement algorithms are measured and reported. Measurement results support program locality and the following working set properties: the average page fault frequency decreases rapidly as τ increases and increases as program size increases. Based on these measured data, working set parameter and process scheduling may be selected from and based on the average page fault frequency. The time slices for the Quantum Queues may be determined from inter-page-fault-time distributions. A page fault scheduling algorithm is proposed for process scheduling in a multi-programming environment. Such an algorithm is effective in reducing CPU idle time and improve system efficiency.

---

*The three measured programs are not short jobs; they should be run in LQQ. Therefore, these measured $P(t,\tau)$'s provide the estimate of time slices for the Long Quantum Queue.

Although the Working Set Algorithm provides an upper bound on replacement algorithm performance, the high cost of implementation prevents it from being widely used. Therefore future research should be in developing low cost hardware devices for economically implementing the Working Set Algorithm or, perhaps even more fruitful, in developing new replacement algorithms that have performance comparable to that of the Working Set Algorithm but are much easier to implement. For example, we have recently studied a Page Fault Frequency Replacement Algorithm. Such an algorithm adjusts the LRU (Least Recently Used) stack according to page fault frequency. Preliminary results already indicate it has excellent performance.

Acknowledgement

The authors wish to thank P.E. Denning of Princeton University for his critical comments on this paper.

APPENDIX

A Cyclic Queueing Model to Study CPU and I/O Operations

To illustrate the relationships among CPU idle time, average page fault frequency and swapping time (time to bring in a new page from the auxiliary memory) T, a cyclic queueing model[8] is used to study CPU and I/O operations. The system in Figure 4 consists of two classes of service facilities. Service facility class I represents a single CPU; its service rate is directly determined by the average page fault rate* λ. Service facility class II represents k parallel I/O servers with each having an average service rate $\mu = \frac{1}{T}$. The k parallel servers represent, for example, a paging drum with k different sectors. Using such I/O facilities, a high degree of overlap of I/O requests can be achieved in a multiprogramming system with relatively low page fault frequency.

Let P_{ij} be the probability that a job leaving server i will proceed to server j. We assume that the job leaves CPU (server 0) and goes randomly to the k I/O servers for service; thus $P_{0j} = \frac{1}{k}$, for $j = 1, 2, \ldots, k$. Since jobs which have finished their I/O operations always return for CPU operations, $P_{i0} = 1$ for $i = 1, 2, \ldots, k$; and all the other P_{ij}'s are equal to zero.

Let N be the total number of jobs in the system, and let n_i denote the number of jobs in service plus the number in queue at the i^{th} server. The state of the system can then be determined by the k + 1 tuple $(n_0, n_1, \ldots, n_k)$ in which $\sum_{i=0}^{k} n_i = N$. The number of distinguishable states of the system--equal to the number of partitions of N customers among k + 1 servers--is $\binom{N+k}{k}$.

*For a system using Working Set Replacement Algorithm with parameter τ, then $\lambda = m(\tau)$.

Let $P(n_0,n_1,\ldots,n_k)$ be the stationary probability that the system is in state $(n_0,n_1,\ldots,n_k)$, and let all the service times be assumed to be exponentially distributed. Then the steady state equations can be written in the form:

$$\left\{\varepsilon(n_0)\ \lambda + \sum_{j=1}^{k} \varepsilon(n_j)\ \mu\right\} P(n_0,n_1,\ldots,n_k)$$

$$= \sum_{j=1}^{k} \varepsilon(n_j)\ \lambda\ P_{0j}\ P(n_0+1,n_1,\ldots,n_j-1,\ldots,n_k)$$

$$+ \sum_{i=1}^{k} \varepsilon(n_0)\ \mu\ P_{i0}\ P(n_0-1,n_1,\ldots,n_i+1,\ldots,n_k) \qquad \text{(A1)}$$

where the indicating function

$$\varepsilon(n_j) = \begin{cases} 0 & \text{if } n_j = 0 \\ 1 & \text{if } n_j \neq 0 \end{cases}$$

accounts for the impossibility of any customer leaving the j^{th} server if that server is empty.

The left hand side of (A1) represents the rate of transition out of state $(n_0,n_1,\ldots,n_k)$; and the right hand side is the rate of transition into this state. Solving (A1) by a method of separation of variables[8], we have

$$P(n_0,n_1,\ldots,n_k) = \frac{1}{G(N)} \prod_{i=1}^{k} \left(\frac{P_{0i}\lambda}{\mu}\right)^{n_i}$$

$$= \frac{1}{G(N)} \left(\frac{\alpha}{k}\right)^{N-n_0} \qquad \text{(A2)}$$

where $\alpha = \lambda/\mu$ and the normalizing function $G(N)$ is determined from the fact that the sum of all the $P(n_0,n_1,\ldots,n_k)$ is equal to 1. Thus

$$G(N) = \sum_{\substack{k \\ \sum_{i=0} n_i = N}} \prod_{i=1}^{k} \left(\frac{\alpha}{k}\right)^{n_i}$$

$$= \sum_{n_0=0}^{N} \binom{N-n_0+k-1}{k-1} \left(\frac{\alpha}{k}\right)^{N-n_0} \qquad \text{(A3)}$$

where $\binom{N-n_0+k-1}{k-1}$ is the number of distinguishable partitions of $N-n_0$ jobs among k I/O servers.

The probability that the CPU is idle is

$$P_0 = \sum_{\substack{k \\ \sum_{i=1} n_i = N}} P(0, n_1, n_2, \ldots, n_k)$$

$$= \frac{1}{G(N)} \binom{N+k-1}{k-1} \left(\frac{\alpha}{k}\right)^{N} \qquad \text{(A4)}$$

For the case k = 1, then (A4) reduces to $P_0 = \frac{\alpha^N}{\sum_{i=0}^{N} \alpha^i}$

For the case N = 3 and k = 6, the values of P_0's for selected α's are shown in Table II.

Table II P_0 vs. α

α	P_0
0.25	0.003
0.50	0.019
1.00	0.091
1.50	0.187
2.00	0.278
2.50	0.362
3.00	0.431
3.50	0.488
4.00	0.537
4.50	0.577
5.00	0.612

We note that α is the ratio of average page swapping time (from secondary memory) to average inter-page-fault-time. A large α implies large page swapping time or small inter-page-fault-time (high page fault frequency), or both. Thus the probability of CPU idle time increases as α increases. Hence, the page fault scheduling algorithm should be effective in reducing CPU idle time and should thus improve system efficiency.

REFERENCES

1. Mattson, R.L. et al., "Evaluation Techniques for Storage Hierarchies," IBM System Journal, Vol. 9, No. 2, pp. 78-117, 1970.

2. Denning, P.J., "The Working-Set Model for Program Behavior," Communications of the ACM, Vol. 11, No. 5, pp. 323-333, May 1968.

3. Denning, P.J. and S.C. Schwartz, "Properties of the Working Set Model," Proceedings of the 3rd ACM Symposium on Operating System Principles, October 1971.

4. Fine, G.H., C.W. Jackson and P.V. McIsaac, "Dynamic Program Behavior Under Paging," Proceedings of the 21st National Conference on ACM, pp. 223-228, 1966.

5. Belady, L.A., "A Study of Replacement Algorithm for a Virtual-Storage Computer," IBM System Journal, Vol. 8, No. 2, 1966.

6. Coffman, E.G. and L.C. Varian, "Further Experimental Data on the Behavior of Programs in a Paging Environment, "Communications of the ACM, Vol. 11, No. 7, pp. 471-474, July 1968.

7. Joseph, M., "An Analysis of Paging and Program Behavior," Computer Journal, Vol. 13, No. 1, February 1970.

8. Gordon, W.T. and G. F. Newell, "Closed Queueing Systems with Exponential Service," Operations Research, Vol. 15, No. 2, April 1967, pp. 245-265.

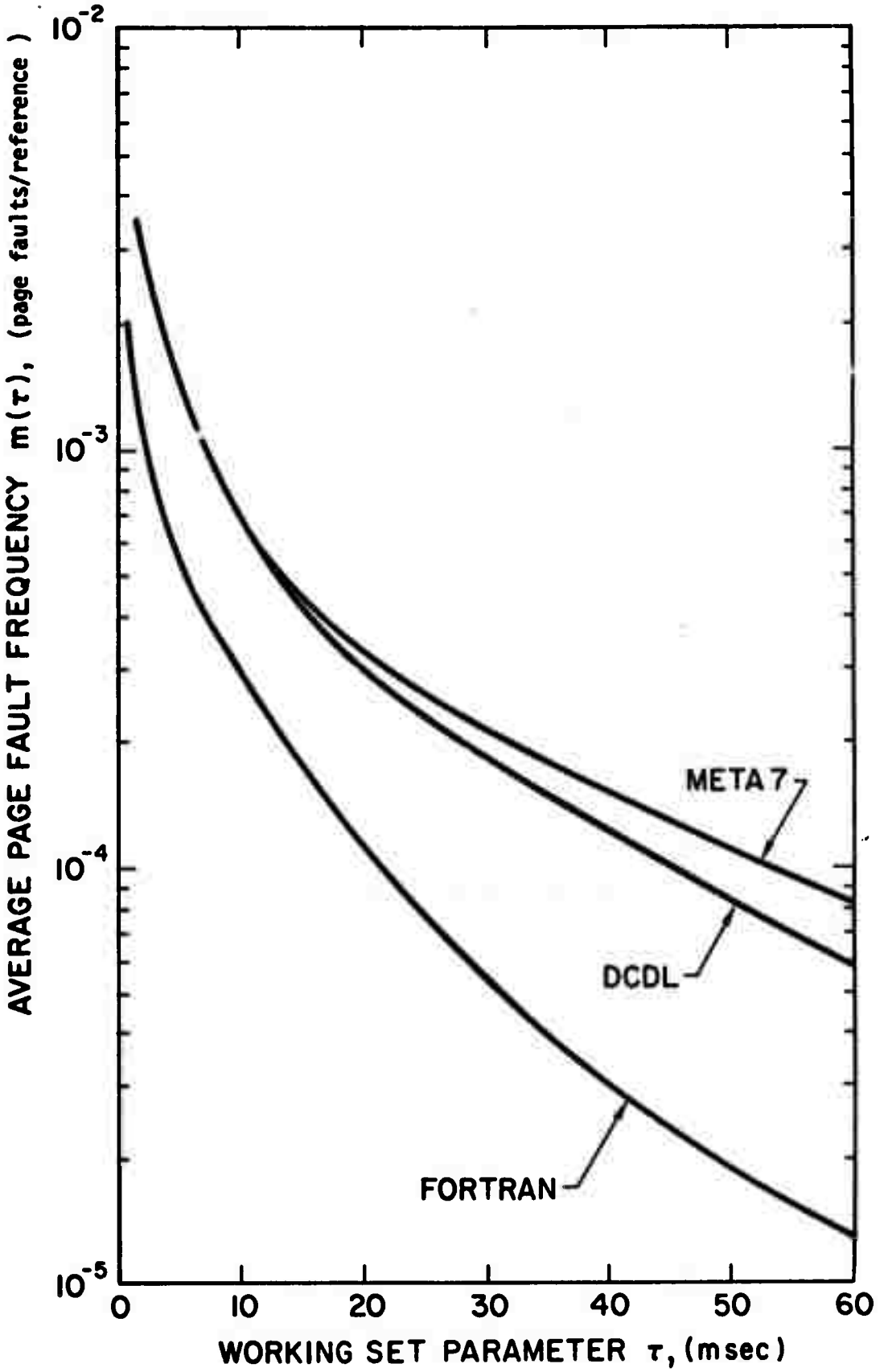


Figure 1. Average page fault frequency m(τ) as a function of working set parameter τ.

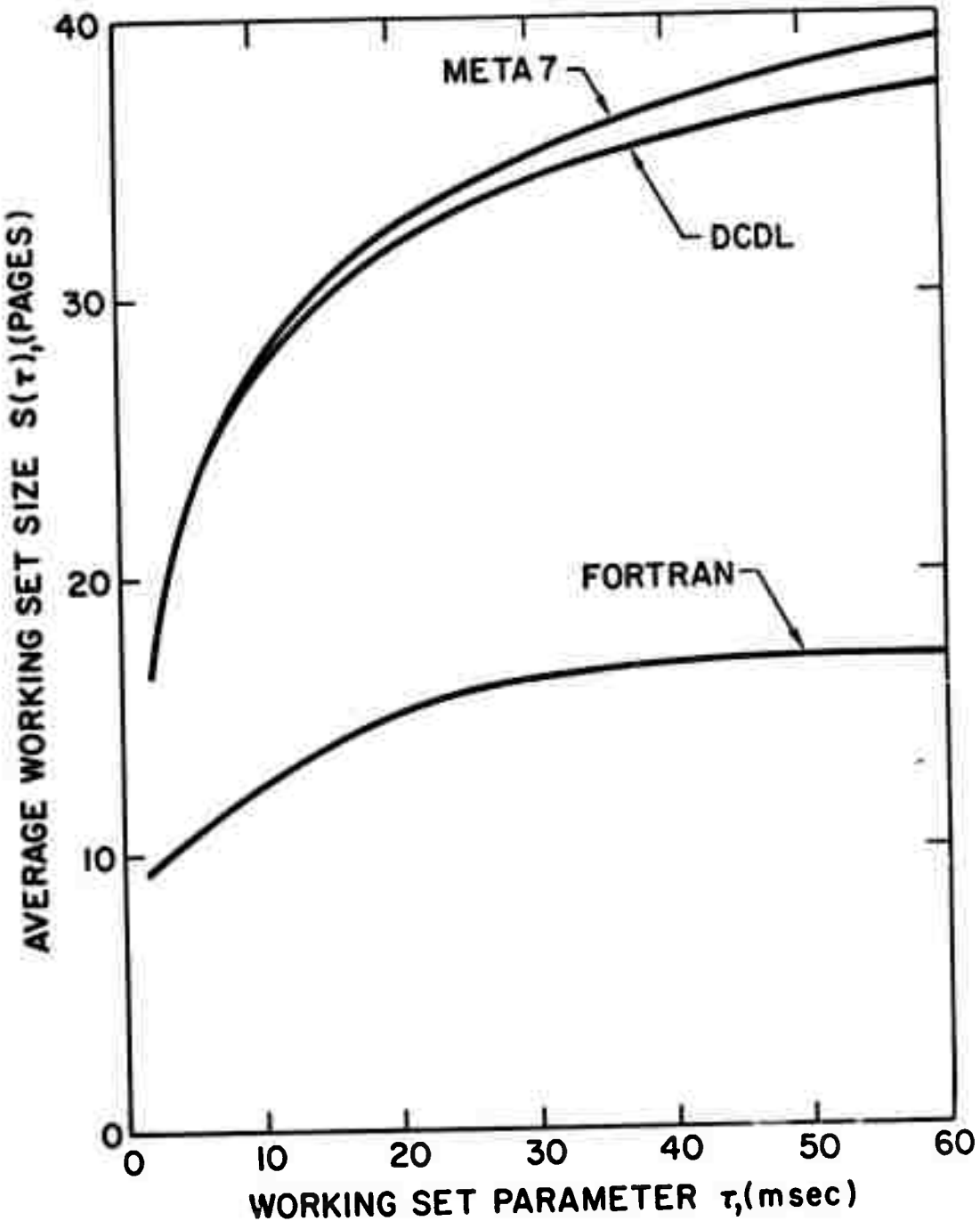


Figure 2. Average working set size $S(\tau)$ as a function of working set parameter τ.

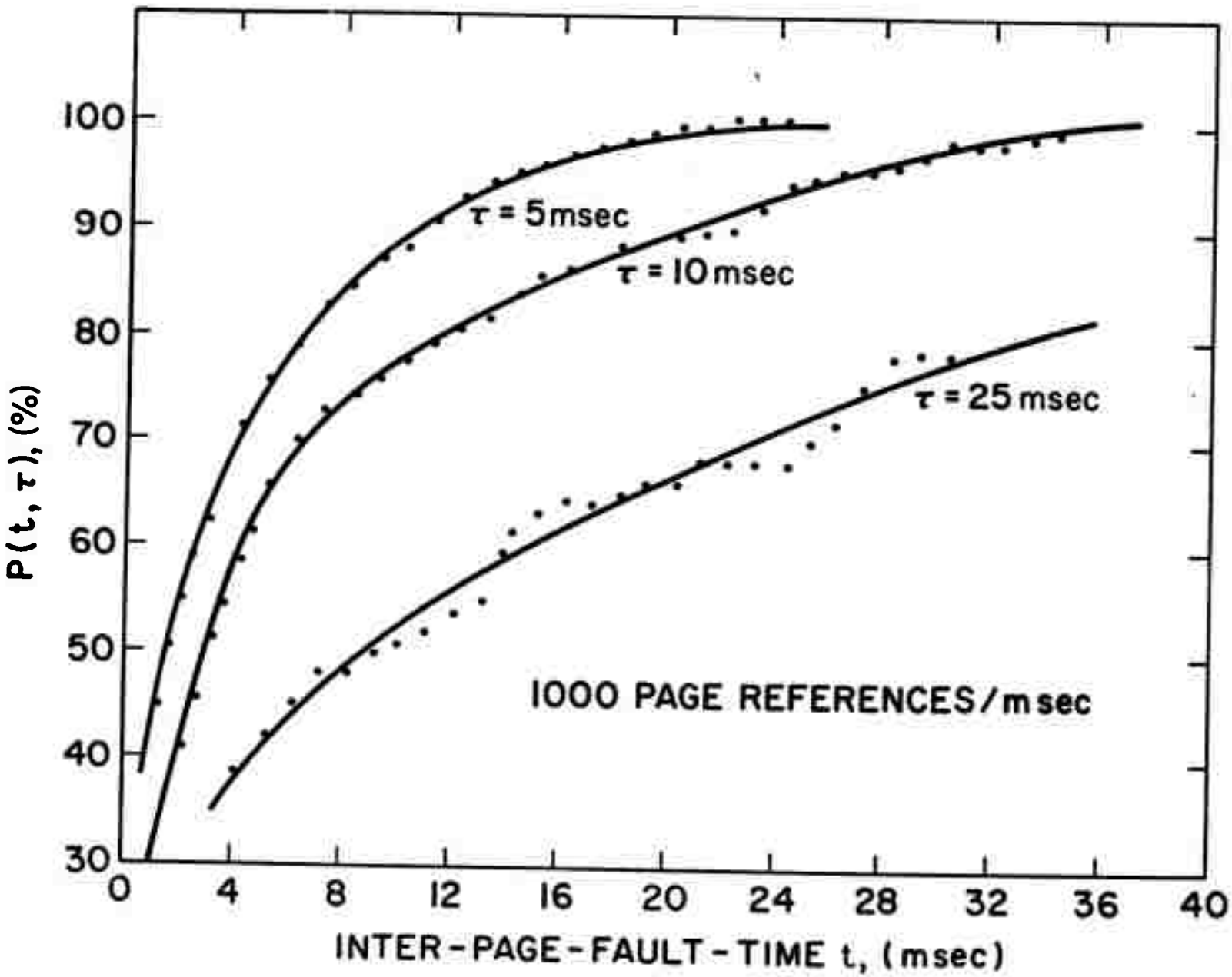


Figure 3. Inter-Page-Fault-Time Distribution
a. Fortran Compiler

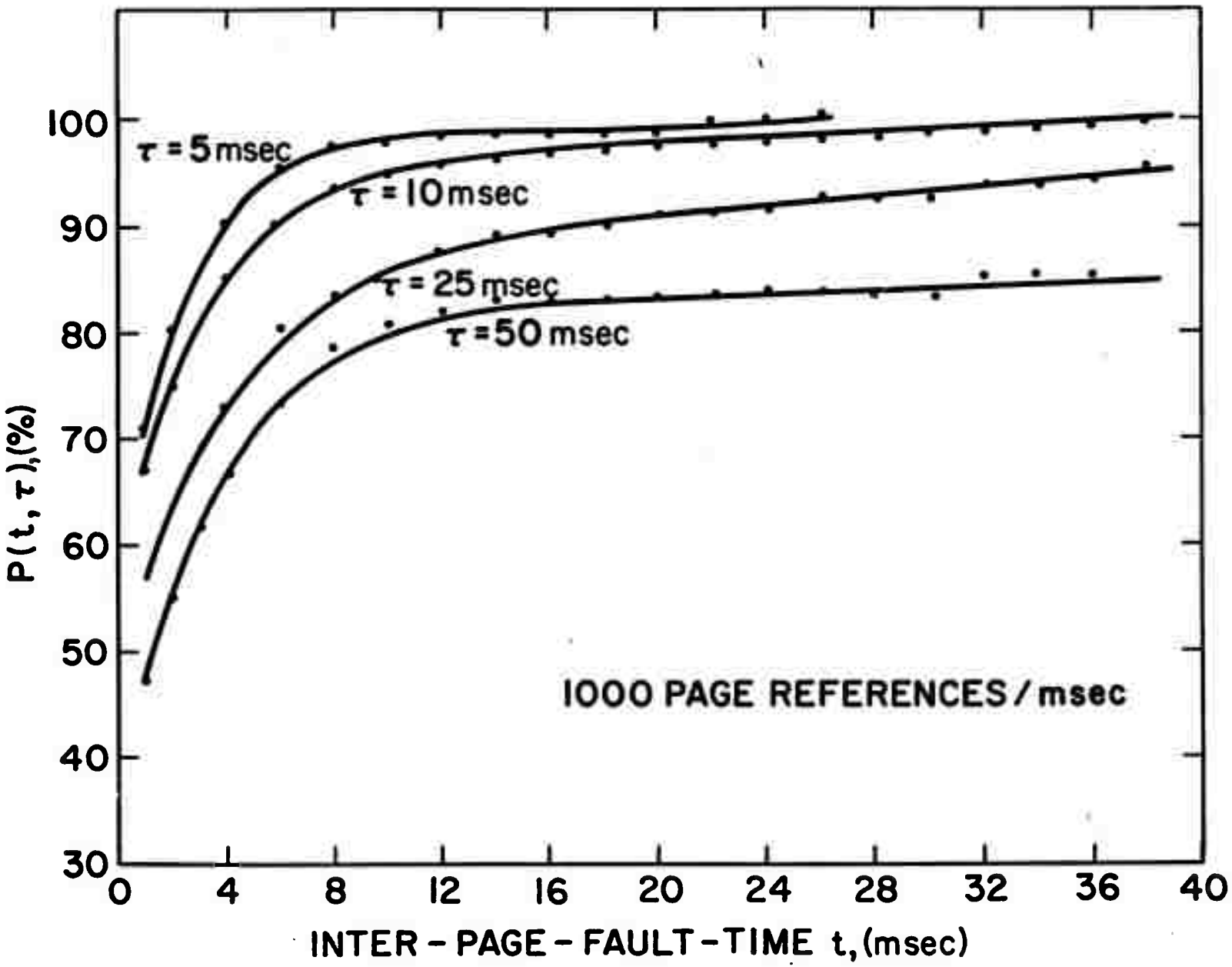


Figure 3. Inter-Page-Fault-Time Distribution
b. DCDL

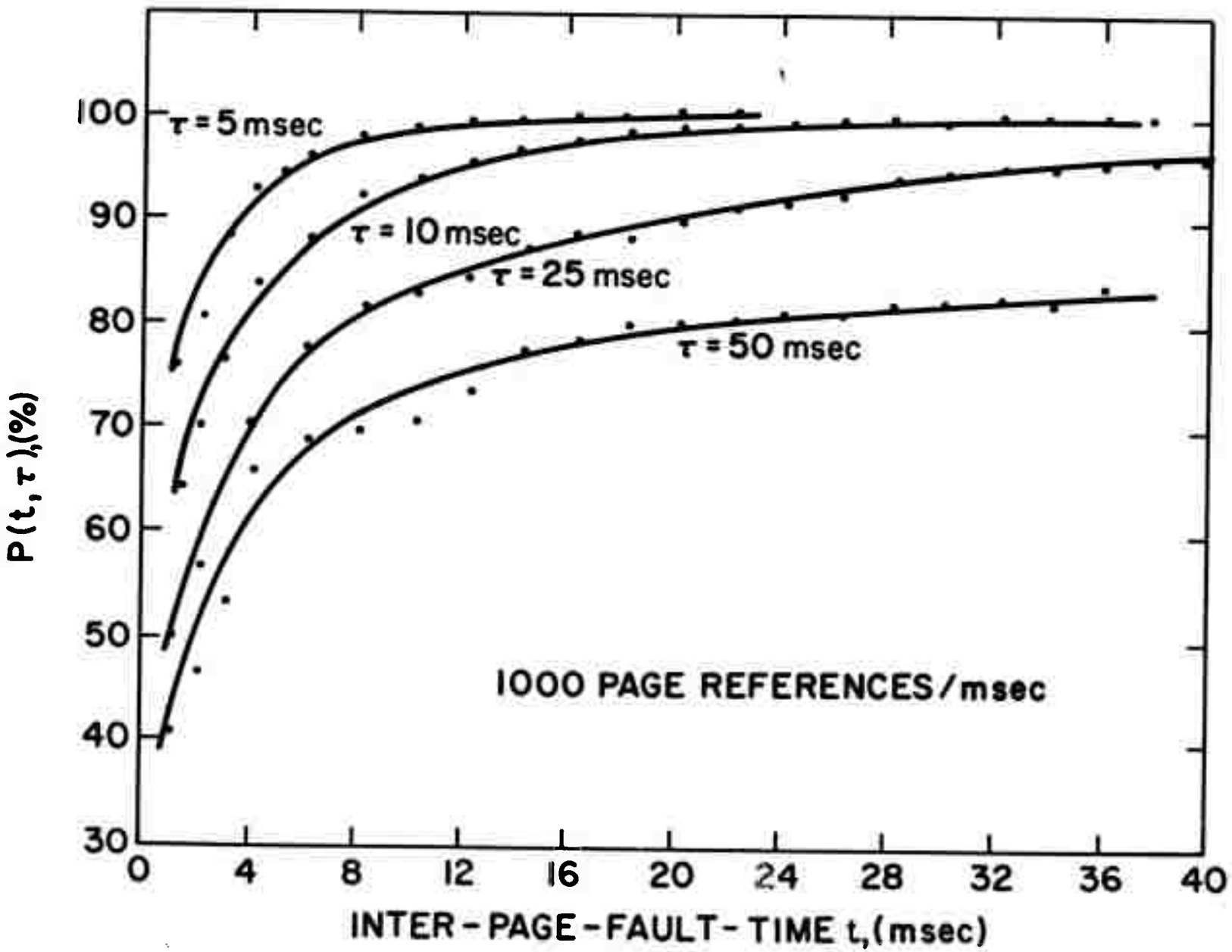


Figure 3. Inter-Page-Fault-Time Distribution
c. Meta 7 Compiler

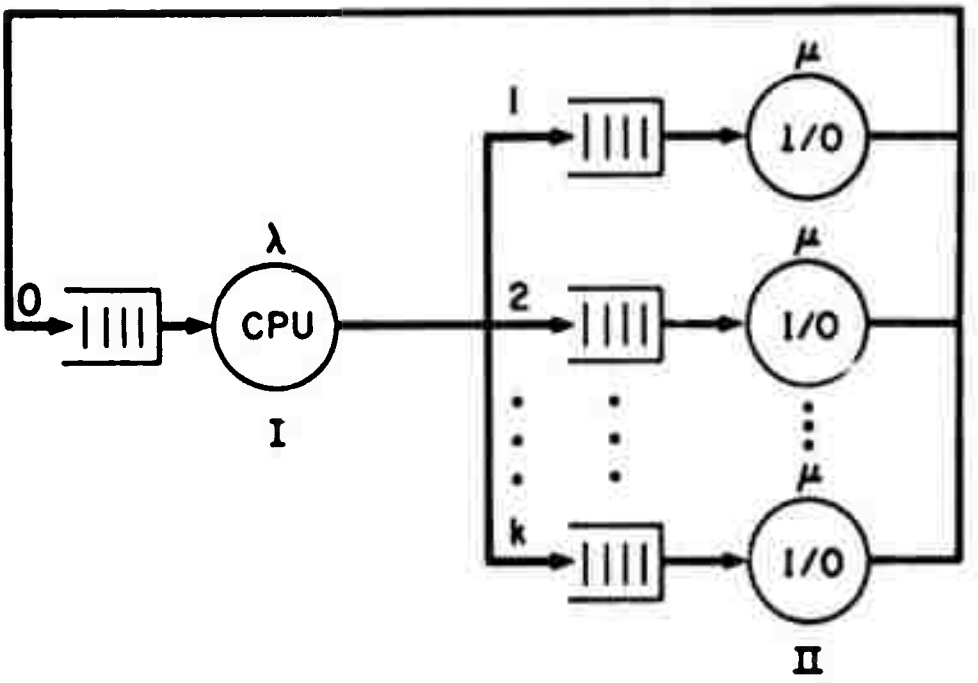


Figure 4. A Cyclic Queueing system for modeling CPU and I/O operations

APPENDIX B

BUFFER BEHAVIOR FOR MIXED INPUT TRAFFIC AND SINGLE CONSTANT OUTPUT RATE

by Wesley W. Chu and Leo C. Liang

Reprinted by permission from
IEEE TRANSACTIONS ON COMMUNICATIONS
Vol. COM-20, No. 2, April 1972

Buffer Behavior for Mixed Input Traffic and Single Constant Output Rate

WESLEY W. CHU, MEMBER, IEEE, AND LEO C. LIANG

***Abstract*—A queueing model with limited waiting room (buffer), mixed input traffic (Poisson and compound Poisson arrivals), and constant service rate is studied. Using average burst length, traffic intensity, and input-traffic mixture rate as parameters, we obtain relationships among buffer size, overflow probabilities, and expected message-queueing delay due to buffering. These relationships are portrayed on graphs that can be used as a guide in buffer design. Although this study arose in the design of statistical multiplexers, the queueing model developed is quite general and may be useful for other industrial applications.**

Paper approved by the Data Communications Committee of the IEEE Communications Society for publication without oral presentation. This research was supported by the U. S. Office of Naval Research, Research Program Office, Contract N00014-69-A-0200-4027, NR 048-129. Manuscript received July 26, 1971; revised October 19, 1971.
The authors are with the University of California, Los Angeles, Calif. 90024.

I. Introduction

In many engineering problems such as computer-storage allocation, data compression, and data communication [1], buffer design is one of the important considerations. Birdsall *et al.* [2], and later Dor [3] have analyzed buffer behavior with Poisson input arrivals and constant output rate. Chu [4] has studied the buffer behavior of a similar model with multiple synchronous constant output rates. He has further studied buffer behavior for batch Poisson arrivals and a single constant output rate [5]. In many data communication systems, input traffic is a mixture of bursts (string of characters) and single characters. For example, in a computer communication system, the cathode-ray-tube terminal outputs are in bursts and the teletypewriter outputs are in characters. Buffer behavior with such mixed input traffic is studied in this paper.

For a given mixed input traffic and a constant output rate, we are interested in 1) the relationship between overflow probability (the average fraction of the total number of arriving characters rejected by the buffer) and buffer size at various traffic intensities, and 2) the expected queueing delay due to buffering. These relationships are obtained by a technique similar to Chu [5]. The results in this paper represent a generalization of his work.

II. Analysis of Buffer Behavior

Let us define the time to transmit a character on the multiplexed line as a unit service interval. The input traffic arriving at the buffer is assumed to be a mixture of single-character inputs and burst (string of characters) inputs. The single-character input X is assumed to be Poisson distributed, with a rate λ_p characters per unit service interval as shown in (1).

$$f_X(k) = \frac{\lambda_p^k}{k!} \exp(-\lambda_p), \qquad k = 0, 1, 2, \cdots. \tag{1}$$

The characteristic function for $f_X(k)$ is

$$\phi_X(u) = \exp[-\lambda_p + \lambda_p \exp(iu)]. \tag{2}$$

For burst input traffic, we assume the length l of the burst Y is geometrically distributed with mean $\bar{l} = 1/\theta$, and the number of bursts Z arriving during a unit service interval is Poisson distributed with a rate λ_c bursts/unit service interval. The distribution of l is

$$f_Y(l) = \theta(1-\theta)^{l-1}, \qquad l = 1, 2, \cdots \tag{3}$$

and the distribution of the number of bursts arriving during a unit service interval is

$$f_Z(n) = \frac{\lambda_c^n}{n!} \exp(-\lambda_c), \qquad n = 0, 1, 2, \cdots. \tag{4}$$

The total number of characters due to burst inputs that arrive during the time to transmit a character on the multiplexed line is a random sum and equals

$$S = \sum_{i=0}^{Z} Y_i \tag{5}$$

where Y_i, a random variable distributed as (3), is the number of characters contained in the ith arriving burst and Z, a random variable distributed as (4), is the total number of

bursts arriving during the unit service interval. All of these random variables are assumed to be statistically independent of each other. It can be shown that $\phi_s(u)$, the characteristic function of S [5], is

$$\phi_s(u) = \exp\{-\lambda_c + \lambda_c\theta \cdot \exp(iu)/[1 - (1 - \theta)\exp(iu)]\}. \quad (6)$$

and $f_s(j)$ (the probability that exactly j characters will arrive due to burst arrivals during a unit service interval) has a compound Poisson distribution

$$f_s(j) = \begin{cases} \sum_{k=1}^{j} \binom{j-1}{k-1} (\lambda_c\theta)^k (1-\theta)^{j-k} \frac{\exp(-\lambda_c)}{k!}, \\ \qquad j = 1, 2, \cdots \\ \exp(-\lambda_c), \qquad j = 0. \end{cases} \quad (7)$$

For mixed traffic of single-character inputs (Poisson) and burst inputs (compound Poisson), the probability that exactly n characters arrive during a unit service interval Π_n is the convolution of $f_x(k)$ and $f_s(j)$; that is,

$$\Pi_n = f_s(n) \circledast f_x(n)$$

$$\Pi_n = \sum_{i=1}^{n} \frac{\lambda_p^{n-i}\exp(-\lambda_p)}{(n-j)!} \sum_{k=1}^{j} \binom{j-1}{k-1} \cdot \frac{(\lambda_c\theta)^k(1-\theta)^{j-k}\exp(-\lambda_c)}{k!} + \frac{\lambda_p^n}{n!}\exp[-(\lambda_p + \lambda_c)], \quad n = 1, 2, \cdots$$

$$\Pi_0 = \exp[-(\lambda_p + \lambda_c)]. \quad (8)$$

The characteristic function for Π_n is

$$\phi_{sx}(u) = \phi_s(u)\phi_x(u) = \exp\{-(\lambda_p + \lambda_c) + \lambda_p \exp(iu) + \lambda_c\theta \exp(iu)/[1 - (1 - \theta)\exp(iu)]\}. \quad (9)$$

The time required to compute Π_n from (8) is dependent on n. For large n (e.g., $n > 1000$), the computation time is prohibitive. Using the same technique as [5], we compute Π_n via the fast Fourier transform (FFT) inversion method as follows:

$$\Pi_n = \frac{1}{M}\sum_{r=0}^{M-1} \phi_{sx}(r)\exp(-2\pi irn/M), \quad n = 0, 1, 2, \cdots, M-1 \quad (10)$$

where

r $2\Pi u/M$;
i $(-1)^{1/2}$;
M total number of points used to represent $\phi_{sx}(r)$ = total number of Π_n.

In order to determine Π_n accurately, they are computed with double precision on the IBM 360/91 at the University of California, Los Angeles. Furthermore, we want to use as many points as possible to represent $\phi_{sx}(r)$; that is, we want to make M as large as possible. Because of the word-length limitation of the computer, double precision provides 15-digit accuracy. Therefore, when $\Pi_n < 10^{-16}$, it is set equal to zero. M is selected such that $\Pi_{n>M} < 10^{-16}$. The M is different for different values of λ_p, λ_c, and l.

Since the buffer has a finite size of N, an overflow will result when a character arrives at the buffer and finds the buffer is full. Thus, the average-character departure rate from the buffer (carried load) β is less than the average-character arrival rate at the buffer (offered load) $\gamma = \lambda_p + \lambda_c l$. The carried load can be computed from the probability that the buffer is busy; that is, $\beta = 1 - p_0$, where p_0 is the probability that the buffer is empty, which can be obtained in the exact manner as in [5].

The traffic intensity ρ measures the degree of congestion and indicates the impact of an input traffic stream upon the departure stream. Since the offered load is represented in a unit service interval, $\rho = \gamma = \lambda_p + \lambda_c l$.

The overflow probability of the buffer (the average fraction of the total number of arriving characters rejected by the buffer) is

$$P_{of} = \frac{\text{offered load} - \text{carried load}}{\text{offered load}} = 1 - \beta/\gamma. \quad (11)$$

Let α be the input-traffic mixture rate that describes the percentage of the traffic contributed by compound Poisson arrivals. Clearly, $1 - \alpha$ is the percentage of the traffic contributed by Poisson arrivals. Thus,

$$\alpha = \lambda_c l/\rho$$

and

$$1 - \alpha = \lambda_p/\rho.$$

In the preceding analysis, we have treated each character as a unit. However, in computing the expected message delay D due to buffering, we should treat each message as a unit. The service time is the time required to transmit the entire message. When the buffer size N is large, for a line with a constant transmission rate, the service-time distribution is the same as the message-length distribution except scaled by a constant transmission rate factor. The message-length distribution for the mixed input traffic of length m is

$$f_M(m) = \frac{\lambda_p}{\lambda_p + \lambda_c}\delta(m) + \frac{\lambda_c}{\lambda_p + \lambda_c}\theta(1-\theta)^{m-1}, \quad m = 1, 2, 3, \cdots \quad (12)$$

where

$$\delta(m) = \begin{cases} 1, & m = 1 \\ 0, & m > 1. \end{cases}$$

When the overflow probability is very small, a good approximation for the expected message delay can be computed from a queueing system with infinite waiting room [4]. The expected queueing delay for an M/G/1 (Poisson arrivals /general service/single output) queueing system is

$$D = \frac{\lambda E[m^2]}{2(1-\rho)} \quad (13)$$

where

$$\lambda = \lambda_p + \lambda_c$$

$E[m^2]$ = second moment of $f_M(m)$.

It can be shown that

$$E[m^2] = \frac{1}{\lambda}[\lambda_p + \lambda_c(2 - \theta)/\theta^2]. \quad (14)$$

Substituting (14) into (13), we have

$$D = \frac{\rho}{2(1-\rho)} + \frac{\alpha(l-1)\rho}{1-\rho}, \quad \text{character-service-times.} \quad (15)$$

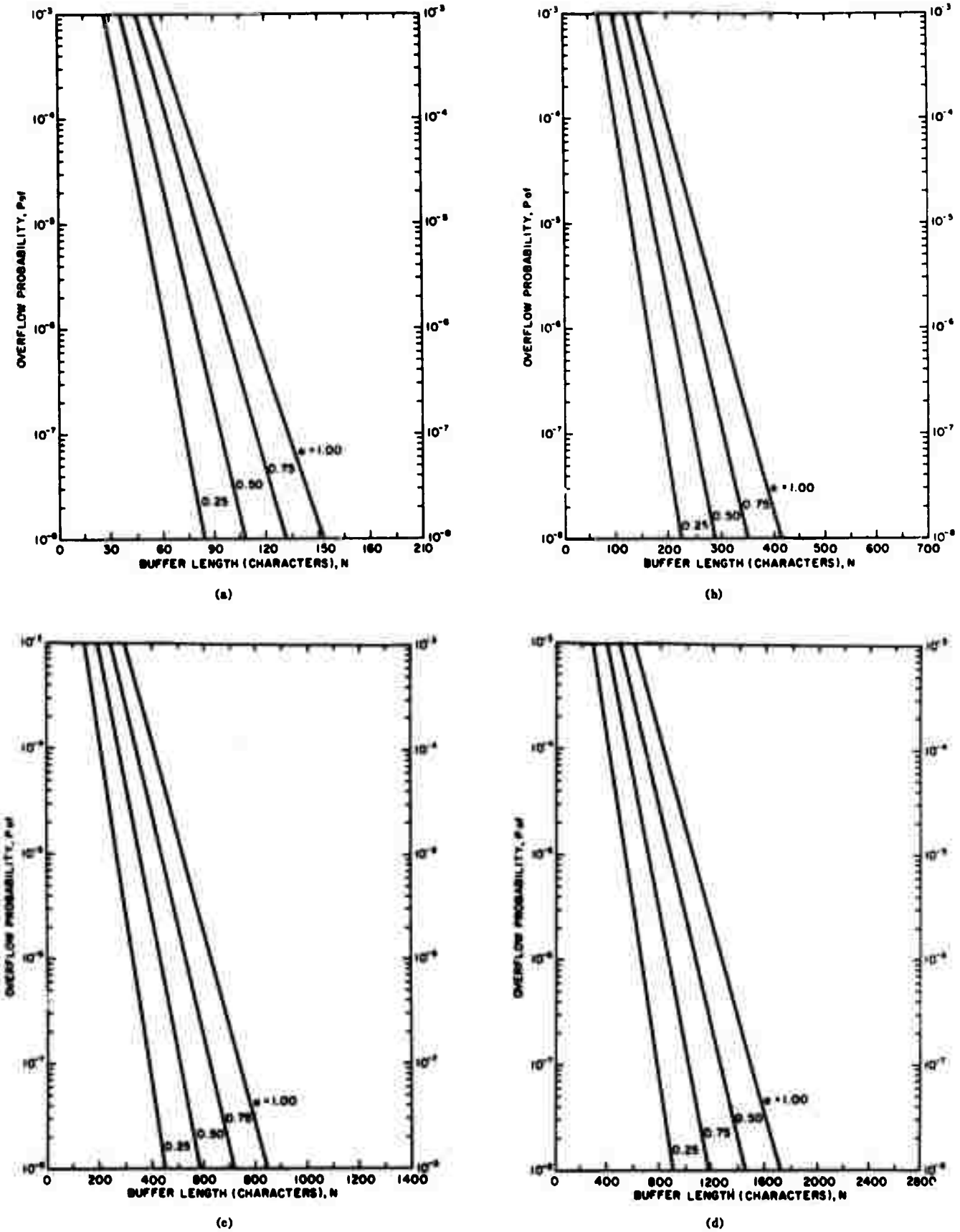


Fig. 1. Buffer length N versus overflow probability P_{of} for traffic intensity $\rho = 0.60$. (a) $\bar{l} = 4$ characters. (b) $\bar{l} = 10$ characters. (c) $\bar{l} = 20$ characters. (d) $\bar{l} = 40$ characters.

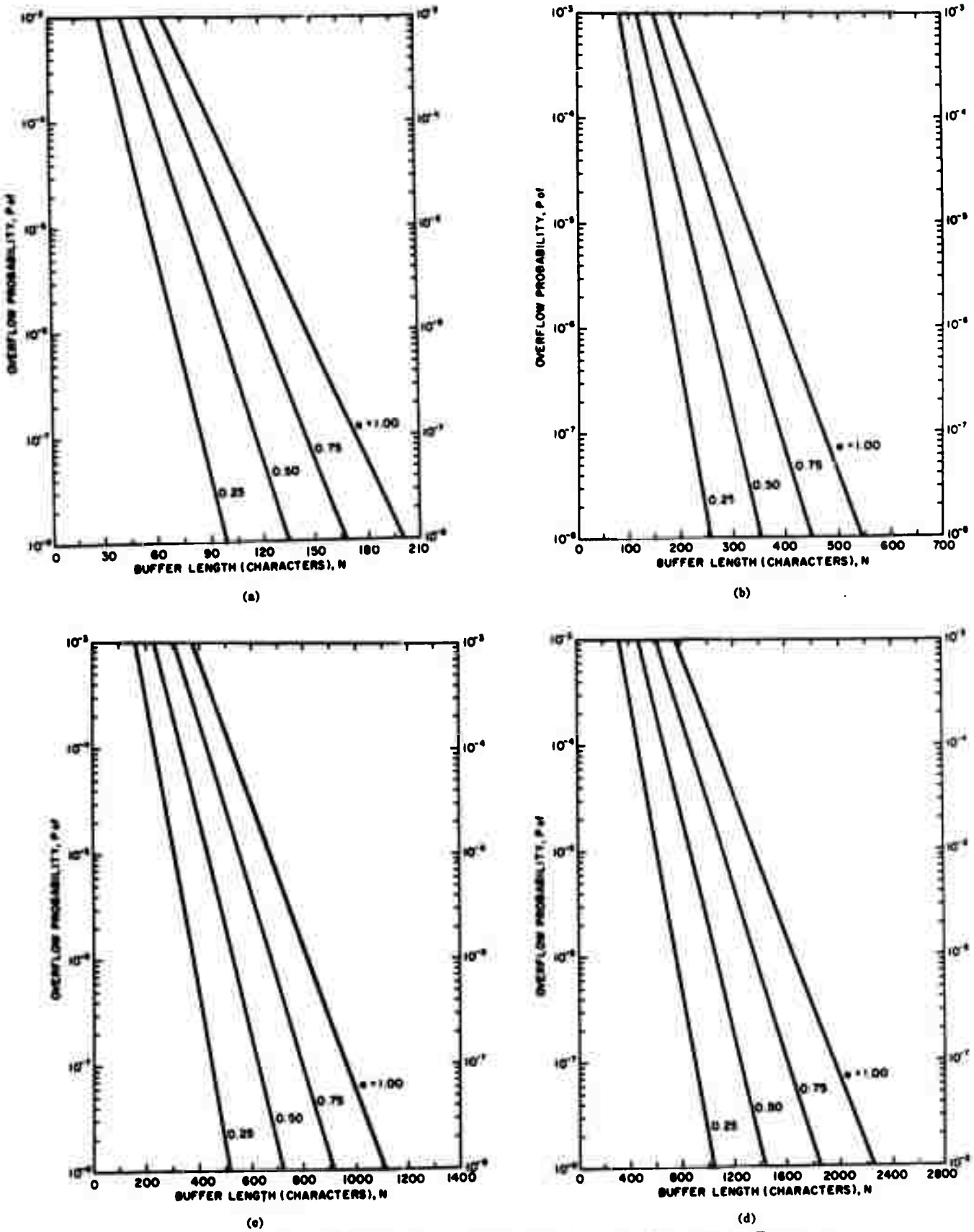


Fig. 2. N versus P_{of} for $\rho = 0.70$. (a) $\bar{l} = 4$ characters. (b) $\bar{l} = 10$ characters. (c) $\bar{l} = 20$ characters. (d) $\bar{l} = 40$ characters.

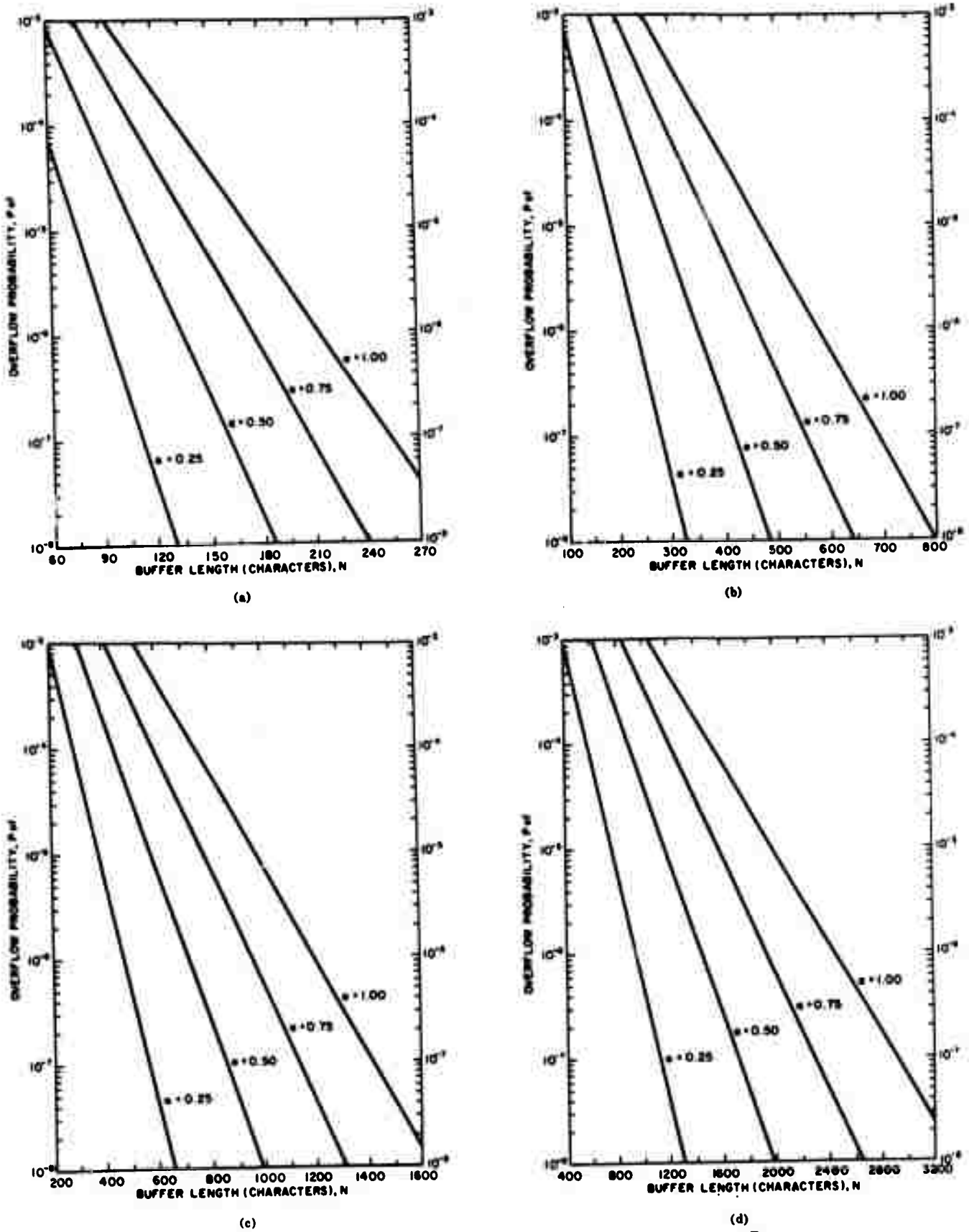


Fig. 3. N versus P_{ovf} for $\rho = 0.80$. (a) $\bar{l} = 4$ characters. (b) $\bar{l} = 10$ characters. (c) $\bar{l} = 20$ characters. (d) $\bar{l} = 40$ characters.

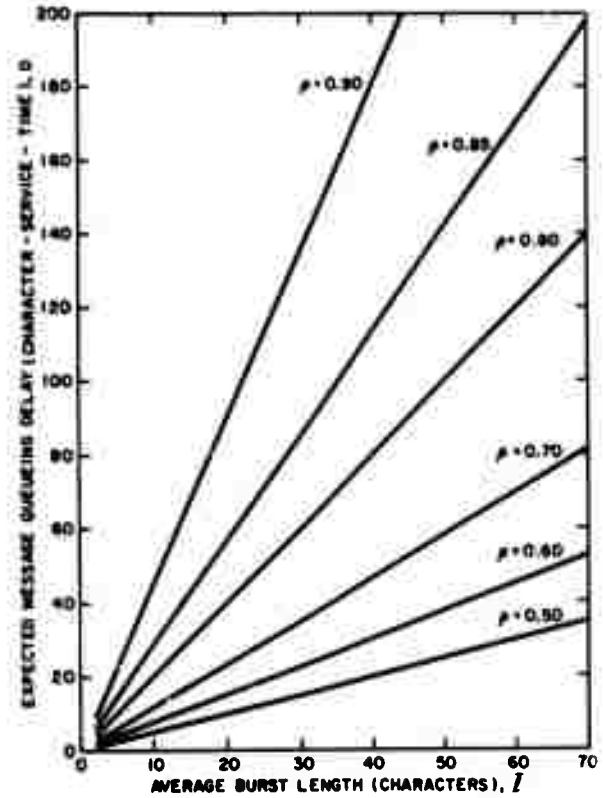


Fig. 4. Average burst length $\bar{l}$ versus expected message delay D for traffic mixture rate $\alpha = 0.5$.

III. Discussion of Results

The relationship of buffer length to overflow probability has been computed for selected traffic intensities ρ, the expected burst length $\bar{l}$, and input-traffic mixture rate α as shown in Figs. 1–3.

The overflow probability depends upon the N, ρ, $\bar{l}$, and α. For a given buffer size N, the overflow probability increases as ρ, $\bar{l}$, and α increase. For a given overflow probability, the required buffer size increases as ρ, $\bar{l}$, and α increase.

In our analysis, we have assumed that the burst length is geometrically distributed and takes values from one to infinity. In practice, however, the maximum burst length is limited to a finite number of characters. Because of the long-tail effect of the geometric distribution, the result obtained here will be more conservative than that of a truncated geometric distribution.

When the average burst length $\bar{l}$ equals unity or when the input-traffic mixture rate α equals zero, the model reduces to the Poisson arrivals with constant output rate, which has been obtained by Birdsall *et al.* [2], Dor [3], and Chu [4]. When α equals unity, then the model reduces to batch Poisson arrivals with a constant output rate, which has been analyzed by Chu [5]. For given ρ and $\bar{l}$, the buffer size required to achieve a desired level of P_{of} is not simply proportional to α. For a desired level of P_{of}, $N\bar{l}$ (the required buffer size for average burst length $\bar{l}$) is much greater than $\bar{l} \cdot N_1$, where N_1 is the required buffer size for burst length of one (Poisson input arrivals).

The expected message delay D due to buffering (calculated from an M/G/1 system with an infinite waiting room) depends upon α, ρ, and $\bar{l}$. For a given ρ and a given $\bar{l}$ (or α), from (15) we note that D is linearly proportional to α (or $\bar{l}$). This agrees with our intuition that for a given traffic intensity, the message delay increases as the message length increases and as the amount of burst input traffic increases. The relationships between $\bar{l}$ and D for $\alpha = 0.5$ and selected ρ are portrayed in Fig. 4.

IV. Conclusion

A finite waiting-room queueing model with mixed input traffic and constant output rate has been studied. For a given traffic intensity and a given input-traffic mixture rate, buffer behavior (in terms of buffer overflow probability and average queueing delay) lies between that for Poisson input arrivals and that for compound Poisson input arrivals. When the traffic mixture rate α approaches zero, the buffer behavior reduces to the Poisson input case. When α equals one, the buffer behavior corresponds to the compound Poisson input case. The numerical results for buffer behavior are portrayed in graphs that are useful in buffer designs.

References

[1] W. W. Chu, "Design considerations of statistical multiplexors," in *Proc. ACM Symp. Problems in the Optimization of Data Communication Systems*, 1969, Pine Mountain City, Ga., pp. 36–60.
[2] T. G. Birdsall, M. P. Ristenbatt, and S. B. Weinstein, "Analysis of asynchronous time multiplexing of speech sources," *IRE Trans. Commun. Syst.*, vol. CS-10, pp. 390–397, Dec. 1962.
[3] N. M. Dor, "Guide to the length of buffer storage required for random (Poisson) input and constant output rates," *IEEE Trans. Electron. Comput.* (Short Notes), vol. EC-16, pp. 683–684, Oct. 1967.
[4] W. W. Chu, "Buffer behavior for Poisson arrival and multiple synchronous constant outputs," *IEEE Trans. Comput.*, vol. C-19, pp. 530–534, June 1970.
[5] ——, "Buffer behavior for batch Poisson arrivals and single constant output," *IEEE Trans. Commun. Technol.*, vol. COM-18, pp. 613–618, Oct. 1970.
[6] E. Fuchs and P. E. Jackson, "Estimates of distributions of random variables for certain computer communication traffic models," *Commun. Ass. Comput. Mach.*, vol. 13, pp. 752–757, Dec. 1970.

APPENDIX C

MODELING, MEASUREMENT AND COMPUTER POWER

by G. Estrin, R. R. Muntz and R. C. Uzgalis

Reprinted from —
AFIPS — Conference Proceedings
Volume 40

Montvale, N. J. 07645

Modeling, measurement and computer power*

by G. ESTRIN, R. R. MUNTZ and R. C. UZGALIS

University of California
Los Angeles, California

INTRODUCTION

Since the early 1960s the literature[9,32] reveals increasing concern with effectiveness of information processing systems and our ability to predict influences of system parameters. A recent survey paper[38] discusses methods of performance evaluation related to three practical goals: selection of the best among several existing systems; design of a not-yet existing system; and analysis of an existing accessible system. The classification of goals is useful, but we can point to neither the models nor the measures nor the measurement tools to allow reliable judgments with respect to those three important goals at this time.

We choose to discuss three issues which do not fall cleanly into Lucas' categories but which are certain to influence our ability to evaluate computer systems in the 1970s. The three issues are: effectiveness of models of computer systems; requirements to be met by measurement experiments; and application of modeling and measurement to the user interface with computer systems.

The first section provides a context for the other sections by reviewing parameters which make computing systems more or less powerful. The second section gives a critique of the state of modeling. The third section characterizes measurement tools. The fourth section discusses the role of measurement at the user interface.

COMPUTER POWER

We consider a computer system to be composed of: a centralized hardware configuration; a set of terminals for entry and exit of user programs and data; an operating system; public programs and data bases; user programs and data; and users and user protocol for entry and exit.

There is no accepted measure for global power or performance of computer systems. There is even no accepted measure for computer cost. Only when a subsystem or subfunction is isolated does it become possible to determine key parameters. However, it is useful to hypothesize such measures and consider influences on them.

Let us, therefore, define a conceptual measure which we call computer system power, P, as a multivariate polynomial function whose coefficients are significance weights. We would, of course, like to have a set of orthogonal functions whose independent variables correspond to measurable parameters but that state of happiness is not apparently within reach. In an attempt to exemplify our philosophy, the authors discuss a set of variables which should influence P keeping in mind that derivation of a figure of merit would require dividing P by some measure of cost.

We intuitively *expect* computer system power to increase if the:

- execution time of any CPU instruction is decreased
- access time of any memory subsystem is decreased
- transfer rate to or from any memory subsystem is increased
- transmission rate of any buss structure is increased
- transfer rate to and from any input or output device is increased
- delay in resource availability is decreased
- error recovery time is decreased
- number of useful public programs is increased
- performance of any public program is increased
- access time to any public data base is decreased
- arrival, execution, and departure rates of user programs are increased
- execution time or resource requirement of any user program is decreased

* This research was supported by the National Science Foundation, Grant No. GJ 809.

- number of effective users increases
- amount of protocol for any user decreases

In a deeper, even more qualitative sense, we expect a computer system to be more powerful if the following conditions hold:

- system manager has a model permitting adaptation to changing load
- errors and system imbalances are reported to maintainers and developers
- program documentation and measurements permit modification with few side effects
- average number of user runs before correct execution is decreased
- the quality of any user program increases in the sense that there is more effective use of a source language on a given computer system.

Although the above observations are useful in stating expected events of concern they ignore interactions between such events and give no indication of weighted importance of the individual events. We further characterize our systems by the following simple remarks.

If the time required for every physical transition to reach its new stable state were halved, we would expect throughput of the system to double. If only some of the events were reduced in transition time, we could no longer guarantee that there would be a reduction in computation time because the scheduling of events is a function of starting and stopping times of concurrent processes. Anti-intuitive anomalies[23,3] are disturbing but do not keep us from conjecturing that they occur only infrequently. If we neglect anomalies, then we cannot expect change in execution time of any one instruction or any one routine or any one compiler to produce a decimal order of magnitude change in a sensibly weighted function of the above parameters. Given reasonable measurement tools and design of measurement experiments we conjecture that somewhere between 10 percent and 50 percent improvement in performance can be accomplished for most systems by changes in assignment and sequencing of resources. Although these percentages do not seem dramatic in their impact, the absolute number of dollars or number of computer hours which would become available is far from negligible.

In contrast with the heuristic probing and tuning of a given system, much greater impact is possible at the user interface with a computer system and by advances in models, particularly validated models of our computer systems. For example, we would guess that there are more than 10 attempts to run a program during its development before it runs once "correctly." For complex programs the ratio of number-of-correct-runs to number-of-runs can approach zero. Hence, if the user interface can be altered so as to increase the probability of a correct run, large benefits may result.

The effect of model development is a more sophisticated and qualitative issue. It is self evident that to the extent that we can predict behavior of even a subsystem through modeling, we can hope to isolate important parameters and the way they affect performance. In fact, only through modeling efforts can we generalize experimental results at one center to apply to many others. Furthermore, it has been recognized that simulation is the most widely used tool in evaluation of systems. If simulation depends upon precise imitation of a computer system, its development cost is generally prohibitive and it is fraught will all the unreliability associated with one-shot development. Effective simulation depends upon validated approximate models of systems and of user programs. Creation of such strong models is the most difficult of our tasks. However, the very process of validating or invalidating simplifying assumptions used in models can lead to new algorithms and improved models. Margolin, Parmelee and Schatzoff[39] very competently demonstrate this effect in their recent study of free-storage management algorithms.

In this section we have taken cognizance of the fact that there is no simple (or even complex) formula for computer performance. The reader's attention has been focussed on the last five in the list of factors affecting computer performance because they offer so much more return. The following sections review work in analytic modeling, measurement, and the user interface.

CRITIQUE OF ANALYTIC MODELING

Any system design, any measurement project or any resource allocation strategy is based on some conception of the environment in which it operates. That conception is a model. It is beneficial to have such models explicitly stated so that they can be explored, tested, criticized and revised. Even better, though not often achieved to the extent desired, is a formal analysis of the models.

Models and methods of analysis vary greatly. Our concern here is with probabalistic models of systems and processes and also with discrete graph models of programs. The goals of these analyses are both insight and quantitative results to influence the design of

systems, resource allocation strategies and possibly the design of languages.

While most will argue that the goals of such analyses are inherently worthwhile and must be pursued, there is widespread dissatisfaction with the current state of the field. Basically, there are three major areas of dissatisfaction. First, the models are generally oversimplified in order to make them mathematically tractable. This obviously makes the results questionable and brings us to the second major failing which is that analytic results are often not validated by measurement or simulation. Moreover, in cases where system evaluation studies are carried out, the existing models do not seem powerful enough to provide a uniform basis for measurements. The third major criticism is that most of the literature on analytic modeling is a collection of analyses of specialized models. This points up the lack of very general powerful results which would allow analysis to become an engineering tool. As it is now, each new situation almost always requires a separate analysis by an expert.

While the above are substantial criticisms, this is not to say that analysis has not had its impact. We can cite, for example, the working set model of program behavior,[12] the work on stack algorithms,[41] studies of time-sharing and multiprogramming system resource allocation and analyses of I/O scheduling,[42,49] the work on data transmission systems and on networks[11,35,19] and the work on graph models of programs.[40,26,27,33,14,24,7,10,22]

Promising areas of research.

Multiple resource models

Much analytic work has dealt with single resource models. The reason for this is clearly that most of the analytic tools which are available apply to single resource environments. The computer system analyst is typically not a mathematician developing new tools but is generally engaged in applying existing tools. Nevertheless, computer systems are multiple resource systems and we must learn to analyze such systems.

Some recent studies of multiple resource models of computer systems have been made using results by Gordon and Newell.[21] The general model considered by Gordon and Newell is one in which customers (or jobs) require only one resource at a time, but move from one resource to another. An example is illustrated in Figure 1 for three resources.

The nodes in this figure represent resources and the arcs represent possible transitions from one resource to another. When a customer has finished at resource i he moves to (requires) resource j next with probability P_{ij}. The arcs are labeled with these probabilities. The service time at each resource is assumed to be exponentially distributed. This is a closed system meaning that the number of customers in the system remains fixed. Gordon and Newell have found expressions for the equilibrium distribution of customers in service or queued at each resource. This allows one, for example, to calculate the utilization of the various resources.

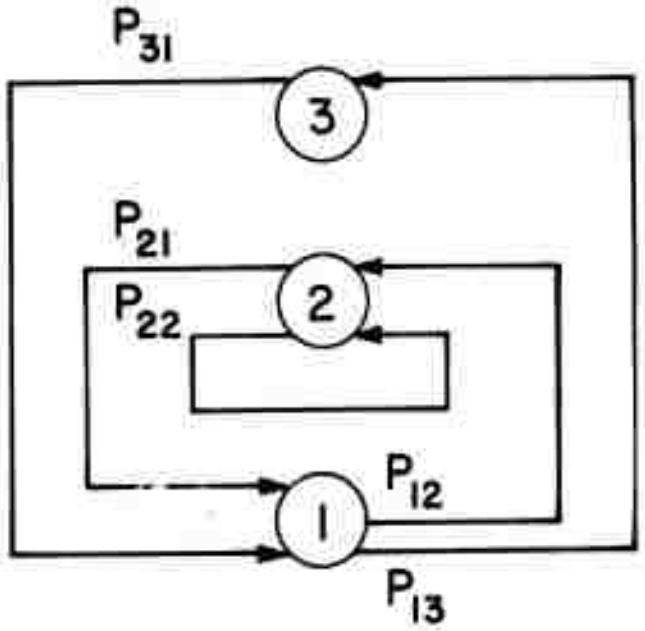


Figure 1—Example network of queues model

Moore[43] and Buzen[8] have applied this model to multiprogramming systems. Moore measured the MTS system to obtain the transition probabilities and mean service times of the resources and then used the model to estimate system parameters such as resource utilizations. The relatively close agreement to measured system parameters leads one to believe that the model can be used to predict the effect of some changes in system configuration. In using the model in this way, one must be careful that the proposed changes do not significantly affect the basic behavior of the customers. Buzen used the same model to gain insight into resource allocation in multiple resource models of computer systems. His studies include the investigation of buffering and the effects of paging algorithms. Both Moore and Buzen have used the model to try to give a meaningful formal definition to the term "bottleneck." It is of interest that they arrive at different definitions of a bottleneck. The reader is referred to the references for details.

While the studies mentioned above are clearly advances in the study of computer system models there are numerous open questions. For example, the model does not allow the representation of the simultaneous use of several resources such as memory and CPU. Also there is no means for representing the synchronization of events such as a process doing buffered I/O. Another limitation is that the customers in the system are assumed to have the same statistical behavior, i.e., the transition probabilities and service time distribution are the same for all customers.

Bounds and approximations

Every evaluation technique makes use of approximations. These approximations may arise, for example: in estimating the system parameters, user and program behavior; or in simplifying the model of the system itself. There is clearly a tradeoff between types of approximations. By simplifying a model one might be able to handle more general types of user and program behavior. Much of the analytic work has been concerned with exact mathematical solutions to models which are themselves gross approximations.

An area which is beginning to be explored is that of approximate solutions to more general models. For example, Gaver has used the diffusion approximation for the analysis of heavily loaded resources in queueing studies.[20] The basic technique is to consider that the work arrival process is not the arrival of discrete customers requiring service but rather a work arrival flow. This work arrival flow is a continuous process with the same mean and variance as the original process. Another example of the use of approximations is the work by Kimbleton and Moore on the analysis of systems with a limiting resource.[24]

It is clear that the use of any approximation requires validation of the results. This may take the form of comparing results with measurements of an actual system, simulation, or obtaining bounds on the error in results. Bounds may also be applied in a different manner. Much has been written on the analysis of time-sharing scheduling algorithms and their effects on response times. Kleinrock, Muntz and Hsu[26] have reported on results which in effect demonstrate the bounds on response time characteristics for any CPU scheduling algorithm which does not make use of *a priori* knowledge of customers service times. The importance of the bounds is that one can see the limits of the variation in response characteristics that are possible by varying the scheduling algorithm and the extent to which these limits have been approached.

Program behavior

A major problem that must be dealt with in any evaluation effort concerned with computer systems is program behavior. Even when using approaches such as benchmarking or trace-driven modeling there is the problem of selection of programs which are in some sense representative of the total population of programs that will be run on the system.

Studies of memory management in particular have had to explicitly include models of program behavior. The early work in this area[2,12] stressed very general but powerful aspects of program behavior such as "locality" and "working set." More recent work deals with more explicit models of the generation of reference strings which assume more about program behavior but correspondingly allow for more detailed analysis.[13] It is hoped that these models will permit more detailed studies of multiprogramming and procedure sharing.

It is interesting to note that the bulk of this work has been directed toward finding models which can represent the universe of possible programs. More particularly, the goals of this research have been to isolate parameters characterizing program behavior to which memory management is sensitive and to compare the effectiveness of various memory management strategies. This approach is in line with a common theme which runs through most of the work on resource allocation strategies in computer systems. That is, we see most allocation strategies attempting to work well over the total population of programs possibly utilizing measurements of recent past history of the process to predict the near future. Outside of work arising from graph models of parallel programs[5,6,51] very little has been done to utilize *a priori* information about a process. Many systems do make *a priori* distinctions between batch and interactive processes. It seems reasonable though that much more information may be available which would be useful in allocating resources. For example, it has been suggested that the time-slice and paging algorithm parameters be tailored to the process.[46] Use of *a priori* information assumes that the process is available for analysis prior to execution. This is a valid assumption for production jobs, system processes, and to some degree for all jobs at compile time. Since these processes consume a significant portion of the system resources, gains in efficiency in managing such processes might result in major gains in total efficiency. There are many open problems associated with this approach:

1. Is there a conflict with a program design goal of program modularity? How is information about separately compiled procedures to be combined?

2. Should processes be permitted to advise the system as to their resource needs? How does the system protect itself against false information?
3. How to manage resources effectively for processes which provide a priori information, and also for processes without associated *a priori* information?
4. What kind of *a priori* information is actually useful to management of a system: how costly is it to obtain and utilize effectively?
5. How predictable are the resource requirements of processes?

While this approach has received only some slight mention in the literature, it appears to be a fertile area for research.

Graph models of programs provide an abstraction of program structure governing flow of control and demand for resources.[21,10,18,22] They permit a representation fitting somewhere between the full detail of actual programs and parametric or stochastic representations of them. Most work using graph models has been concerned with *concurrent processing*. However, the graph model analyses explicitly reveal sets of independent tasks which become candidates for alternate sequencing *in sequential systems*.

Ideally, we search for models of systems and program behavior which provide principles guiding synthesis of configurations along with well founded resource management strategies. Measurement must validate effectiveness of such strategies. The diversity of computations further demands that measured parameters be provided to operating systems and users in order to permit adaptation to dynamic variations in system behavior and to unavoidable anomalies in systems and languages.

Studies during the latter half of the '60s showed how little attention had been given to measurability in the man-made universe of computer systems. The next section characterizes some of the problems in measurement.

MEASUREMENT OF INFORMATION PROCESSING SYSTEMS

Tools for measurement of computer systems must satisfy all of the following requirements: detection of prescribed events; recording of detected events; retrieval of accumulated records; data reduction; and display.

We comment on each in turn.

Detection

We start by rejecting the absurdity of observing all of the states of a system under observation since it would imply detecting the state of every input, every memory element and every output every time there was a change, along with the time at which the change occurred. Hence, any set of measurement tools must include means of selecting a subset of system states.

Hardware measurement tools provide a prescribed number of sensing probes which may be physically placed on selected register or buss points in a machine under observation. Measurement system registers along with programmed comparators and basic logical operations permit further filtering by allowing detection of a subset of the events sensed by the probes. Even with such filtering the rate of change of detected states may be excessive. If the response time of hardware measurement elements is insufficient, basic circuit changes would be required to make the measurement feasible. If bandwidth is insufficient, it is sometimes possible to introduce a sampling signal and thereby further reduce the number of detected events. *In the absence of interaction with software monitor programs, a hardware monitor is clearly limited in its utility.* To be convinced of this, one need only consider the kind of program status information change which is observable by probes only when it appears in the form of an operand during the course of computation. Hardware detection can have the virtue of introducing no artifact into the measured system and of being able to detect events whose states are not accessible to measurement programs. Sampled detection may be made more effective by allowing interference with the observed process. If a sampling signal enforces a proper interruption, observed data may be sequentially sensed by detection circuits. The recently reported "Neurotron" monitor[1] is the most interesting implemented hardware monitor, and its design shows the foresight of enabling interaction with software monitor programs.

Software measurement tools consist of programs which detect selected events by virtue of their insertion at state-change points in the sequential computational process.[17,47,31] This detection process introduces artifact in execution time, in space required for measurement program storage, and sometimes (e.g., synchronization with asynchronous cyclic processes) in qualitative side effects on existing computational processes. In a sampling mode, measurement programs can have their in-line artifact reduced by disturbing the flow of computation only at a sampling time. At a sampling time, measurement programs may be brought in to check as large a set of memory states as is needed and

then control is returned to the observed system. In the absence of hardware support, a software monitor is limited to observation of those system states which have affected memory contents. In one case, careful analysis of measurement of PL/I functions of an IBM 360/91[10] revealed anomalies in recorded system states which can best be characterized as artifact introduced by OS/360 when it inserts code associated with I/O interrupts into the code being measured.

It has become clear that we are not faced with mutually exclusive alternatives of hardware detection tools or software detection tools. Rather how much of each; how they are integrated; and how they are made available to experimenters. A paper by Nemeth and Rovner[45] presents a pleasing example of the power of combined hardware and software in the hands of a user. They point out that facilities introduced for hardware debugging are often the kind useful in later program measurements.

Recording

If an event of interest has been detected, its occurrence must affect memory contents. Such action may be as simple as incrementing a counter or as complex as storing a lot of state information for later analysis. In the case of nondisturbing hardware measurements, external storage must be provided and the transfer rate must be able to keep up with the rate of change-of-state information observed by a set of probes and associated circuits. In the case of software measurements, sufficient memory space must either be provided to record all relevant state information, or else preprocessing reduction programs must be called in to reduce storage requirements.

Retrieval

In the construction of any large system, both a data gathering and retrieval system must be incorporated into the basic design. Failure to do so will limit the amount of instrumentation available later when efficiency questions arise. For example, in a large programming system which has transient program segments, data gathering is easily inserted into any program segment; however, unless a standard data storing program is available, the data gathered cannot be easily retrieved. The IBM PL/I F-level compiler is an example of a programming system broken into transient program segments. It fails to have a data storing program with adequate bandwidth to support meaningful measurement activity.

Data Reduction

The amount and kind of data reduction is determined by the goal of the measurement experiment and limitations of measurement tool capabilities in detection, recording and preparation for retrieval. For example, assume that we want to obtain a history of utilization of routines in order to decide which should be kept in primary storage and which should be kept in backup storage. Assume, further, that every time a routine is called: the name of the called routine, the time of day, and the name of the user is recorded. It would not be very meaningful to generate only a history showing the times at which each routine in the system was used by each user. Data reduction would be required to determine, for example, the total number of such uses, an ordering of routines by number of uses, a determination of the number of routines involved in, say 50 percent, of the uses and their names, etc.

Display

The goal of the data reduction process is defined by the specified form of display or feedback to the experimenter. If measurement is being made for feedback to an operating system for use in resource allocation, parameter values must be delivered to memory cells to be accessed by the operating system. If measurement is made for accounting purposes or, more generally, to provide the user with feedback about his quality of use and system response, results should be merged into user and system manager files. If measurement is made for operator control and management, simple alphanumeric displays are common. For experimental analysis of system behavior, CRT displays, graphs and computer printout are generally required.

Measurement Methodology

The complexity of computer systems dictates special care in the planning of measurement experiments. If the results of experiments are not reproducible, they are of little value. If any assumptions made are not recorded and validated, the results cannot be generalized and applied at another time or place. A large body of statistical theory is available providing methods for abstracting properties out of individual data points, but applicability must be carefully checked. We have little hope of adhering to principles if we do not have a measurement language to prescribe measurement experiments as sequences of commented operations which are appropriately integrated with observed data. The latter step needs wait upon creative development of

measurement tools and their test in meaningful experiments. Measurement capability must be explicitly included during periods of system design and must be available for inclusion in user program design. Digital computer systems and programs are properly characterized as complexes of very elementary functions. Full systems or programs, therefore, generally require partition in order to manage the synthesis process. Each partition introduces possible measures of validity of output, of performance and of cost. Means for measurement should be checked at that point in design and a value judgment made if excessive artifact would be introduced by the measurement process.

If a system contains the structure and primitive operations satisfying the five requirements discussed in this section, it carries the tools for adaptability. We conjecture that much more than the 10 percent to 50 percent improvement alluded to in the Introduction becomes attainable—particularly when measurement tools can influence user behavior.

COMPUTER POWER AND USER INTERFACE

In the seventies some stronger effort must be directed toward increasing computer power by a reduction of the complexity of the user interface.

Operating systems and Higher Level Languages are tools designed to give the user control of the portion of a computer system he needs. With the exception of work done at SDC,[28,29] little reported effort has been devoted to the human engineering aspects of these tools. During the last decade, while hardware made a dramatic increase in power, the management tasks required of the operating system increased from trivial to highly complex. At the same time, the users were required to supply (in unnatural form like JCL) more of the parameters which would allow effective management decisions to be made by the operating system. These user-supplied parameters have increased the burden of complexity at the user interface—and reduced the amount of useful work a user can accomplish in a given period of time.

For example, much of the attraction of APL/360 is its simplification of the operating system interface along with the addition of immediate execution of its concise powerful primitives. A batch oriented FORTRAN user perceives this as a tremendous increase in his computer power. A more sophisticated user might see APL as a powerful desk calculator which provides immediate access to functions similar to functions he already commands, less accessibly, in other languages.

Another user interface problem exists at the level of higher level languages. As more advanced hardware becomes available to the user, he seeks to solve more complex problems. When a problem grows beyond a manageable point, the user segments the problem into pieces plus associated linkages. In doing so, however, he introduces a new set of communication problems; a change in one program which affects an interface can now wreak havoc in another "completed" portion of the problem solution. Higher level languages have been lax in the types of program interconnections (and interactions) allowed.

An example of the problems of creating a large programming system are reported by Belady and Lehman using data from the development of OS 360.[4] While this study concerned programs written in assembly language for the IBM 360, the properties which produce the error rates and modification ratios reported in their paper are characteristics of all large programming systems today.

Several techniques for improving the probabilities that a program can be made error free are available in the literature. One of the earliest is Dijkstra's "Notes on Structured Programming,"[16] and also "THE Programming System."[15] His system breaks the problem into shells of "pearls" or "primitive" operations. Each shell is built using the "primitives" of the next lower level. This system attempts to minimize interactions, forces the programmer to produce generalized functions which can be tested, and allows easy instrumentation because of the segregation of functions.

Some disadvantages of such a hierarchical scheme make its practical application difficult. Such a scheme increases initial development time because it forces developers to completely understand the structure of the system being built and to predefine the proper function for each hierarchy. Transitions between levels may be costly. Functions at the lowest level are the most general and, therefore, the most frequently used. Small inefficiencies in these functions, or in the method of traversing levels of the structural hierarchy, magnify costs dramatically and force the user away from centralized functions. This defeats the original purpose of the organization.

Another disadvantage of a hierarchical scheme is that while instrumentation of the system is easy, interpretation of the measurements is generally not. Measurement results could change drastically if the organization of the program were modified. Therefore, it is hard to tell how much of what goes on is due to the structural hierarchy and how much is due to the intrinsic properties of the program. Such knowledge points a way toward improvement.

	NUMBER OF ERROR OCCURRENCES	ERROR TYPE*	ERROR DESCRIPTION
COMPILE-TIME			
	263	IEM0227	NO FILE/STRING SPECIFIED. SYSIN/SYSPRINT HAS BEEN ASSUMED.
	87	IEM0182	TEXT BEGINNING yyyy SKIPPED IN OR FOLLOWING STMT NUMBER
	74	IEM0725	STATEMENT NUMBER xxxx HAS BEEN DELETED DUE TO A SEVERE ERROR NOTED ELSEWHERE.
	63	IEM0152	TEXT BEGINNING yyyy IN STATEMENT NUMBER xxxx HAS BEEN DELETED.
	46	IEM1790	DATA CONVERSION WILL BE DONE BY SUBROUTINE CALLS.
	39	IEM0185	OPTION IN GET/PUT IS INVALID AND HAS BEEN DELETED.
	27	IEM0677	ILLEGAL PARENTHESIZED LIST IN STATEMENT NUMBER xxxx FOLLOWS AN IDENTIFIER WHICH IS NOT A FUNCTION OR ARRAY.
	27	IEM0109	TEXT BEGINNING yyyy IN OR FOLLOWING STATEMENT NUMBER xxxx HAS BEEN DELETED.
	25	IEM0096	SEMICOLON NOT FOUND WHEN EXPECTED IN STATEMENT xxxx. ONE HAS BEEN INSERTED.
	23	IEM0673	INVALID USE OF FUNCTION NAME ON LEFT HAND SIDE OF EQUAL SYMBOL OR IN REPLY, KEYTO OR STRING OPTION.
EXECUTION-TIME			
	14	IHE804	ADDRESSING INTERRUPT.
	12	IHE320	FIXED OVERFLOW.
	8	IHE140	FILE name—END OF FILE ENCOUNTERED.
	7	IHE604	ERROR IN CONVERSION FROM CHARACTER STRING TO ARITHMETIC.

* IBM, PL/I(F) *Programmers' Guide* (*Appendix K*), GC28-6594, January 1971.

Figure 2a—Most frequent PL/I errors

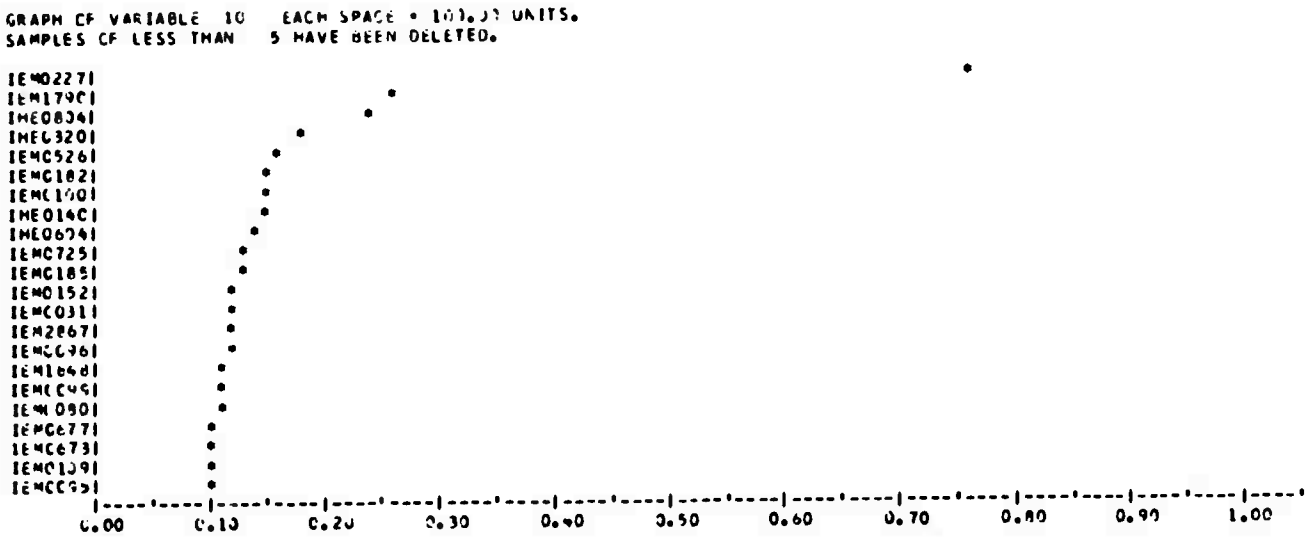


Figure 2b—Average persistence sorted by average persistence

	NUMBER OF ERROR OCCURRENCES	ERROR TYPE*	ERROR DESCRIPTION
COMPILE-TIME			
	143	SY16	IMPROPER ELEMENT(S)
	131	SYE5	ILLEGAL USE OF COLUMN 1 ON CARD
	89	CGOC	NO FILE SPECIFIED. SYSIN/SYSPRINT ASSUMED
	83	SY0B	MISSING SEMICOLON
	76	SM4E	ident HAS TOO MANY SUBSCRIPTS. SUBSCRIPT LIST DELETED
	55	SY3A	IMPROPER LABEL
	53	SY04	MISSING)
	48	SY06	MISSING COMMA
	46	SY09	MISSING :
	40	SM50	name NEVER DECLARED, OR AMBIGUOUSLY QUALIFIED (EXPRESSION REPLACED OR CALL DELETED)
EXECUTION-TIME			
	827	EX78	SUBSCRIPT number OF ident IS OUT OF BOUNDS
	239	EX83	FIXED POINT OVERFLOW
	211	EXBB	DELETED STATEMENT ENCOUNTERED
	189	EX7D	LENGTH OF SUBSTRING LENGTH OF STRING
	180	EX98	INCOMPATABLE OPTIONS ON OPEN
	169	EX7B	INDEX OF SUBSTRING LENGTH OF STRING
	141	EXB8	ARRAY ELEMENT HAS NOT BEEN INITILIZED. IT IS SET TO 0.
	90	EX9F	IMPLIED CONVERSION NOT IMPLEMENTED
	46	EX89	PROGRAM IS STOPPED. NO FINISH ON-UNIT AFTER ERROR
	45	EXB7	ident HAS NOT BEEN INITIALIZED.

* Conway, R. W. et al,. *User's Guide to PL/C, The Cornell Compiler for PL/I*, Release 6, Department of Computer Science, Cornell University, Ithaca, August 1, 1971.

Figure 3a –Most frequent PL/C errors

Present studies of forced program structure and program proofs of correctness may begin to provide models on which HLL designers may base their proposals. However, any major changes should be designed to improve the user's system so that each program submittal can be a learning experience. In this way a programming system can be called upon to point out unusual events; draw the programmer's attention toward possible errors; and yet, not produce volumes of output which would be costly to print and which a programmer would refuse to read.

Work at Cornell toward producing compilers which correct human failings rather than punish them has culminated in a highly functional, rapid compiling, and very permissive PL/I student-oriented compiler called PL/C.[44] This compiler does spelling correction of keywords, automatic insertion of missing punctuation, etc. In addition automatic collection of some statistics is done at execution time. For example, each label causes a count to be maintained of the number of times execution passed through that labeled statement. Compilers such as these increase computer power by reducing the complexity of the user interface.

Implementations of HLLs could further help a programmer by giving an optional cross reference listing showing locations where a variable is changed, could be changed, or just referenced. Items could be flagged if they were never referenced or set; only referenced; or only set. In the first two cases spelling correction might be applicable. Statements which use expensive library subroutines or other costly language features could be flagged. Measurement nodes could be easy to insert and operate. These should, in turn, produce meaningful data which relate directly to questions the programmer wanted to ask.

But such discussions have only beat around the bush itself. The real problem, the bush, is the higher level language. The real questions are: What features are error prone? What features of the language allow automatic validity checking of what is written? How can these properties be identified and measured? How can the knowledge of these things be used to reduce complexity of the user interface so that the user perceives an increase in his computer power? Which language constructs are seldom used, adding unnecessary complexity and unreliability?

Efforts to measure the human engineering aspects of computer language use[37] and to provide feedback

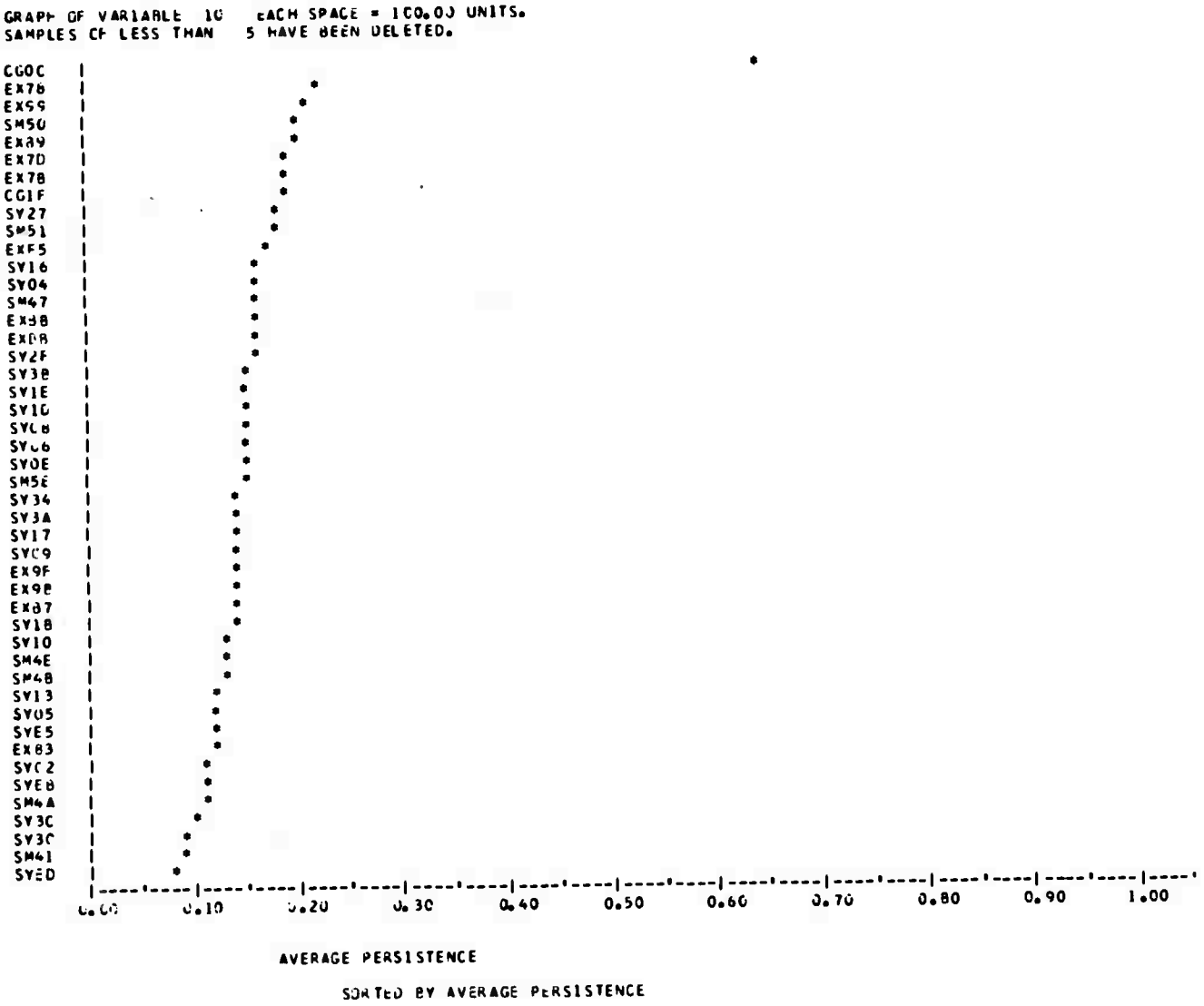


Figure 3b

into the design stages of higher level languages and the control and command languages of the operating system may provide major increases in computer power by:

- increasing the number of users who can bring problems to the machine
- decreasing the number of problem submissions necessary to bring a job to completion

Work is in progress at SDC,[28,29,48] moving toward a man-machine symbiosis. These little publicized approaches to measurement of human problem solving and computer languages have just begun to scratch the surface of this very important area. Work at UCLA has attempted to identify properties of PL/I which are prone to human error. As a first approximation, error rates and persistence curves of various errors identified in students' use of the IBM PL/I F-level compiler[30] is presented in Figure 2. Corresponding results for errors found by Cornell's PL/C compiler are presented in Figure 3. Figure 2a shows a table of the number of occurrences of the most frequent PL I error types recorded during compilation and during execution times. Figure 2b displays the persistence of errors by PL/I type during the student runs. The vertical coordinate is the error type ordered by the magnitude of PERSISTENCE RATIO. The hori-

zontal coordinate is the PERSISTENCE RATIO and was calculated as an average of (number of sequential trials during which the particular error persisted) divided by the total number of trials. If an error type did not occur in at least 5 problem assignments it was arbitrarily deleted to keep the displayed range of values reasonable. Figures 3a and 3b display the same properties for assignments using PL/C. A total of 128 problem assignments completed by 28 students are included in the statistics. Follow-up work is intended to lead more deeply into language design and hopefully into new techniques for automatically localizing errors in a program.

The basic technique for doing this is to allow the programmer to specify more information than the HLL processor needs to compile the program. An example would be identifiers of the form "CONSTANT" in a PL/I data attribute syntax. CONSTANTs as opposed to variables, would only be set by an initial attribute and would be illegal as a left-hand side of an assignment or as an argument of a pseudo-variable. In addition, the program could be considered as having this attribute. At several points in a program (e.g., block exit time) these constants would be checked to see if their value had changed. If any had, a warning would be printed; the correct value restored; and the program would continue. Such a new PL/I data type allows automatic checking for consistency to localize errors and yet is almost painless for a programmer to use. When a program is debugged, it is easy to turn off this kind of checking for the sake of more efficient performance. In a hardware environment like the MULTICS GE 645, these errors can be detected dynamically when illegal accesses occur.

Debugging tools should be designed into the language and taught as part of the language, because the majority of the time a programmer deals with a language, he is also dealing with an incorrect program. Subscript checking, trace information, validity checking at periodic intervals, time and count information, formatted displays of all program information and selective store and fetch recording are the kinds of things which should be available to the HLL programmer.

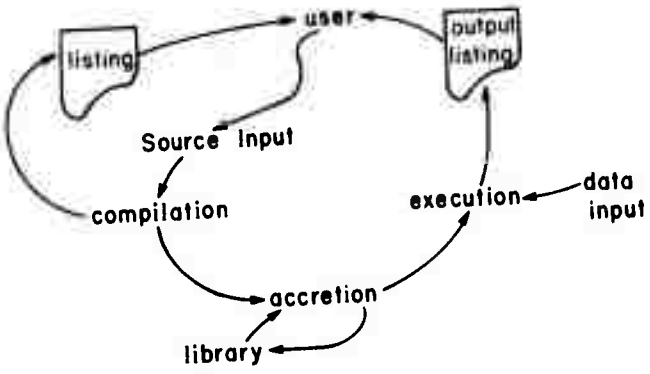


Figure 4a—Information flow in a standard HLL job

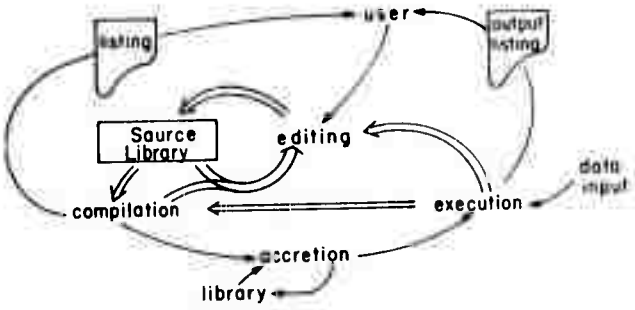


Figure 4b—Modified information flow to augment feedback in a HLL job

In addition, measurement tools should be immediately accessible to any user without burdening others, so that if questions of efficiency are raised they can be answered simply and quickly. Some of the measurement tools which seem important are: (1) flow charts or tables as optional output which would stress intermodule dependencies; (2) time and count control statements which could be output, reset and inactivated under program control, and would create output automatically if the program terminated abnormally or without expressly outputting the data gathered; (3) program size should be easily accessible by the program dynamically and summaries of available space should be available at program termination.

In order to increase the complexity of the programming problems which users can handle, languages must be allowed to accommodate personal ways of expressing concepts. To do this, at the very minimum, new data types should be available for the programmer to define as well as operators which use these data types. This begins to syntactically approach the Dijkstra concepts and to allow easier application of hierarchically structured programs. Hopefully these approaches will increase the user's computer power by making the development of his programs easier.

The programming system itself should be restructured so that more information is available to a HLL processor. Figure 4a shows the diagram of information flow in a usual batch oriented system. The source code is compiled; the resulting object code is passed to the accretion step where library or previously compiled programs are added to it; and the resulting load module

is passed to the execution phase; finally, output from execution is passed back to the user. To provide more automated feedback, Figure 4b shows information flow in a system where statistical summaries of one execution are available to the next compilation. In addition, the accretion step spends much more of its time checking linkage conventions and the validity of argument-parameter choices. This programming system has an edit/compile time library which is designed to help make changes easy. For example, it keeps "COMMON" declarations uniform (read EXTERNAL if you are a PL/I programmer) and it also uses information from the compiler to point the user at syntax errors and information from the execution phase to point the user at semantic errors.

Such modifications can reduce errors and speed the development of programs by improving communication between what are now considered separate program steps. However, the most important changes, across all the proposed modifications, are those changes which will allow the programmer to receive only those pieces of information relevant to the level at which he is programming (i.e., making changes). This would provide dynamic help; help where the programming language acts as an extension of the users' mind to assist in problem solving and optimization.

It is important to view these changes which move toward dynamic assistance in terms of costs. Each change must cost something in execution time overhead. Some of the more powerful features like selective fetch and store monitoring must be expensive. However, if these features were found valuable, then modification to hardware might diminish costs dramatically. Integrating these techniques into HLLs must be inherently costly because implementation and testing of human interaction with these diagnostic features are difficult to execute in any controlled way—much work must rest on subjective evaluation of users' behavior. Integration of aids into HLL translators must be initially done without those very aids which are deemed necessary to help programmers modify programs. Therefore, any change is fraught with risks caused by lack of checks in current systems. Obviously bootstrapping is called for and we can expect many passes before achieving effective tools.

The development of richer higher level languages on the one hand, and the development of debugging services and error correcting compilers on the other, exert forces in the direction of increasing performance at the user interface. With appropriate use of models and measurement much more improvement may be obtained.

SUMMARY

Computer systems are different from other systems by virtue of the dynamic fashion in which our comprehension of their behavior may be built into their operation. If validated models are developed, they may then be built into the system itself to serve adaptive resource allocation algorithms. If measurement tools are effectively integrated, they may be made available to the user to improve the quality of his use of programming languages. If the user is, in fact, a team developing programming systems, the modeling and measurement facilities may serve to make much more complex programs possible because a model of programs being built is, itself, generally too complex for a group of unaided humans to manage in an error free way. In the above paper we have sought to open up these questions.

We have not had much experience with effective modeling and measurement. There is an immense amount of data to be observed in a computer system. Cost-effectiveness of performance measurement must be considered. As one of our reviewers put it, "This reviewer has seen some measurement studies lead to system improvements which will pay off sometime in 2018." Hopefully the 1970s will see more effective modeling and measurement introduced into the design process and selectively carried into developed systems to help both internal process management and the enrichment of external use through the user interface.

REFERENCES

1 R A ASCHENBRENNER L AMIOT
N K NATARAJAN
The neurotron monitor system
AFIPS Conference Proceedings Vol 39 pp 31-37 1971 Fall Joint Computer Conference Las Vegas Nevada November 1971

2 L A BELADY
A study of replacement algorithms for a virtual storage computer
IBM Systems Journal Vol 5 No 2 pp 78-101 1966

3 L A BELADY R A NELSON G S SHEDLER
An anomaly in the space-time characteristics of certain programs running in paging machines
Communications of the ACM Vol 12 No 6 pp 349-353 June 1969

4 L A BELADY M M LEHMAN
Programming system dynamics or the meta-dynamics of systems in maintenance and growth
IBM Report No I.C 3546 September 1971

5 D P BOVET G ESTRIN
A dynamic memory allocation in computer systems
IEEE Transactions on Computers Vol C-19 No 5 pp 403-411 May 1970

6 D BOVET G ESTRIN
On static memory allocation in computer systems
IEEE Transactions on Computers Vol C-19 No 6 pp 492-503 June 1970

7 T H BREDT
Analysis of parallel systems
IEEE Transactions on Computers Vol C-20 No 11 pp 1403-1407 November 1971

8 J P BUZEN
Queueing network models of multiprogramming
PhD Dissertation Harvard University Cambridge Mass August 1971

9 V G CERF
Measurement of recursive processes
Computer Science Department Technical Report No UCLA-ENG-70-43 University of California Los Angeles May 1970

10 V G CERF K GOSTELOW S VOLANSKY E FERNANDEZ
Formal properties of a graph model of computation
Computer Science Department Technical Report No UCLA-ENG-7178 December 1971

11 W W CHU
A study of asynchronous time division multiplexing for time sharing computers
AFIPS Conference Proceedings Vol 39 pp 669-678 1971 Fall Joint Computer Conference Las Vegas Nevada November 1971

12 P DENNING
The working set model for program behavior
Communications of the ACM Vol 5 No 11 pp 323-333 May 1968

13 P DENNING S C SCHWARTZ
Properties of the working set model
Proceedings of the Third Symposium on Operating Systems Principles pp 130-140 Stanford University October 1971

14 J B DENNIS
Programming generality, parallelism and computer architecture
Proceedings of the IFIP Congress 68 Booklet C Software 2 pp C1-C7 North Holland Publishing Co Edinburgh England August 1968

15 E W DIJKSTRA
The structure of the THE multiprogramming system
Communications of the ACM Vol 11 No 5 pp 341-346 May 1968

16 E W DIJKSTRA
Notes on structured programming
Report No EWD249 Technische Hogeschool Eindhoven Spring 1969

17 G ESTRIN D HOPKINS B COGGAN S D CROCKER
SNUPER COMPUTER—Instrumentation automation
AFIPS Conference Proceedings Vol 30 pp 645-656 1967 Spring Joint Computer Conference Atlantic City New Jersey April 1967

18 E B FERNANDEZ
Restructuring and scheduling of parallel computations
Presented at the Fifth Asilomar Conference on Circuits and Systems Pacific Grove California November 8-9 1971 To be published in the proceedings of that conference

19 H FRANK I T FRISCH W CHOU
Topological considerations in the ARPA computer network
AFIPS Conference Proceedings Vol 3 pp 581-587 1970 Spring Joint Computer Conference Atlantic City New Jersey May 1970

20 D P GAVER
Diffusion approximations and models for certain congestion problems
Journal of Applied Probability Vol 5 pp 607-623 1968

21 W J GORDON G F NEWELL
Closed queueing systems with exponential servers
ORSA Journal Vol 15 No 2 pp 254-265 March 1967

22 K GOSTELOW
Flow of control, resource allocation, and the proper termination of programs
Computer Science Department Technical Report No UCLA-ENG-7179 University of California Los Angeles California December 1971

23 R L GRAHAM
Bounds for certain multiprocessing anomalies
The Bell System Technical Journal pp 1563-1581 November 1966

24 A N HABERMANN
Prevention of system deadlocks
Communications of the ACM Vol 12 No 7 pp 373-377 July 1969

25 L E HART G J LIPOVICH
Choosing a system stethoscope
Computer Decisions Vol 3 No 11 pp 20-23 November 1971

26 A W HOLT H SAINT R M SHAPIRO S WARSHALL
Final report for the information system theory project
Rome Air Development Center by Applied Data Research Inc Contract No AF30(602)-4211 1968

27 A W HOLT F COMMONER
Events and conditions
Parts 1-3 Applied Data Research Inc 450 Seventh Avenue New York New York 10001 1969

28 A HORMANN A LEAL D CRANDELL
User Adaptive Language (UAL): A step towards man-machine syncrgism
SDC TM-4539 June 1969

29 A HORMANN S KAUFMANN-DIAMOND C CINTO
Problem solving and learning by man-machine teams
SDC TM-4771 July 1971

30 *PL/I(F) compiler program, logic manual*
GY28-6800 Fifth Edition IBM 1966, 67, 68, 69 October 1969

31 D H H IGNALLS
FETE—A FORTRAN Execution Time Estimator
Computer Science Department Report No STAN-CS-71-204 Stanford University February 1971

32 R R JOHNSON
Needed: A measure for measure
Datamation pp 22-30 December 15 1971

33 R M KARP R E MILLER
Parallel program schemata
Journal of Computer and System Sciences Vol 3 No 4 pp 147-195 May 1969

34 S R KIMBLETON C G MOORE
A limited resource approach to system performance evaluation
Technical Report No 71-2 Department of Industrial Engineering University of Michigan June 1971

35 L KLEINROCK
Analytic and simulation methods in computer network design
AFIPS Conference Proceedings Vol 36 pp 569-579 1970 Spring Joint Computer Conference Atlantic City New Jersey May 1970

36 L KLEINROCK R R MUNTZ J HSU
Tight bounds on the average response time for time-shared computer systems
Proceedings of the IFIP Congress 71 TA-2 pp 50-58 Ljubljana Yugoslavia August 1971

37 D E KNUTH
An empirical study of FORTRAN programs
Computer Science Department Report No CS-186 Stanford University 1971

38 H C LUCAS JR
Performance evaluation and monitoring
ACM Computing Surveys Vol 3 No 3 pp 79-91 September 1971

39 B H MARGOLIN P P PARMELEE M CHATZOFF
Analysis of free-storage algorithms
IBM Systems Journal Vol 10 No 4 pp 283-304 1971

40 D F MARTIN
The automatic assignment and sequencing of computations on parallel processor systems
PhD Dissertation and Computer Science Department Technical Report No UCLA-ENG-66-4 University of California Los Angeles 1966

41 R MATTSON J GECSEI D SLUTZ I TRAIGER
Evaluation techniques for storage hierarchies
IBM Systems Journal Vol 9 No 2 pp 78-117 1970

42 J M McKINNEY
A survey of analytical time-sharing models
Computing Surveys Vol 1 No 2 pp 105-116 June 1969

43 C G MOORE
Network models for large-scale time-sharing systems
PhD Dissertation University of Michigan April 1971

44 H MORGAN R A WAGNER
PL/C—The design of a high performance compiler for PL/I
AFIPS Conference Proceedings Vol 38 pp 503-510 1971 Spring Joint Computer Conference Atlantic City New Jersey May 1971

45 A G NEMETH P D ROVNER
User program measurement in a timed-shared environment
Communication of the ACM Vol 14 No 10 pp 661-666 October 1971

46 J RODRIGUEZ-ROSELL
Experimental data on how program behavior affects the choice of schedular parameters
Proceedings of the Third Symposium on Operating Systems Principles pp 156-163 Stanford University October 1971

47 E C RUSSELL G ESTRIN
Measurement based automatic analysis of FORTRAN programs
AFIPS Conference Proceedings Vol 34 pp 723-732 1969 Spring Joint Computer Conference Boston Massachusetts May 1969

48 H SACKMAN
Man-computer problem solving
Auerbach Publishers Inc 1970

49 T J TEOREY T B PINKERTON
A comparative analysis of disk scheduling policies
Proceedings of the Third Symposium on Operating Systems Principles pp 114-121 Stanford University October 1971

50 R C UZGALIS J ALLEN
360 model 91 execution times of selected PL/I statements
Modeling and Measurement Note No 7A September 1971
360 assembly language source code for selected PL/I statements with model 91 execution times
Modeling and Measurement Note No 7B September 1971
Computer Science Department University of California Los Angeles

51 S A VOLANSKY
Graph model analysis and implementation of computational sequences
PhD Dissertation and Computer Science Department Technical Report No UCLA-ENG-70-48 University of California Los Angeles 1970

APPENDIX D

COMPUTER COMMUNICATION NETWORK DESIGN -- EXPERIENCE WITH THEORY AND PRACTICE

by H. Frank, R. E. Kahn and L. Kleinrock

Reprinted from —
AFIPS — Conference Proceedings
Volume 40

Montvale, N. J. 07645

Computer communication network design— Experience with theory and practice*

by HOWARD FRANK
Network Analysis Corporation
Glen Cove, New York

ROBERT E. KAHN
Bolt Beranek and Newman Inc.
Cambridge, Massachusetts

and

LEONARD KLEINROCK
University of California
Los Angeles, California

INTRODUCTION

The ARPA Network (ARPANET) project brought together many individuals with diverse backgrounds, philosophies, and technical approaches from the fields of computer science, communication theory, operations research and others. The project was aimed at providing an efficient and reliable computer communications system (using message switching techniques) in which computer resources such as programs, data, storage, special purpose hardware etc., could be shared among computers and among many users.[38] The variety of design methods, ranging from theoretical modeling to hardware development, were primarily employed independently, although cooperative efforts among designers occurred on occasion. As of November, 1971, the network has been an operational facility for many months, with about 20 participating sites, a network information center accessible via the net, and well over a hundred researchers, system programmers, computer center directors and other technical and administrative personnel involved in its operation.

In this paper, we review and evaluate the methods used in the ARPANET design from the vantage of over two years' experience in the development of the network. In writing this paper, the authors have each made equal contributions during a series of intensive discussions and debates. Rather than present merely a summary of the procedures that were used in the network design, we have attempted to evaluate each other's methods to determine their advantages and drawbacks. Our approaches and philosophies have often differed radically and, as a result, this has not been an easy or undisturbing process. On the other hand, we have found our collaboration to be extremely rewarding and, notably, we have arrived at many similar conclusions about the network's behavior that seem to be generally applicable to message switched networks.

The essence of a network is its design philosophy, its performance characteristics, and its cost of implementation and operation. Unfortunately, there is no generally accepted definition of an "optimal" network or even of a "good" network. For example, a network designed to transmit large amounts of data only during late evening hours might call for structural and performance characteristics far different from one servicing large numbers of users who are rapidly exchanging short messages during business hours. We expect this topic, and others such as the merits of message switching vs. circuit switching or distributed vs. centralized control to be a subject of discussion for many years.[1,14,24,32,34,37]

A cost analysis performed in 1967-1968 for the ARPA Network indicated that the use of message switching would lead to more economical communications and better overall availability and utilization of resources than other methods.[36,38] In addition to its impact on the availability of computer resources, this decision has generated widespread interest in store-and-forward communications. In many instances, the use of store-and-forward communication techniques can result in

* This work was supported by the Advanced Research Projects Agency of the Department of Defense under Contract No. DAHC 15-70-C-0120 at the Network Analysis Corporation, Contract Nos. DAHC 15-69-C-0179 and DAHC-71-C-0088 at Bolt Beranek and Newman Inc., and Contract No. DAHC 15-69-C-0285 at the University of California at Los Angeles.

greater flexibility, higher reliability, significant technical advantage, and substantial economic savings over the use of conventional common carrier offerings. An obvious trend toward increased computer and communication interaction has begun. In addition to the ARPANET, research in several laboratories is under way, small experimental networks are being built, and a few examples of other government and commercial networks are already apparent.[6,7,31,40,41,47,48,52]

In the ARPANET, each time-sharing or batch processing computer, called a Host, is connected to a small computer called an Interface Message Processor (IMP). The IMPs, which are interconnected by leased 50 kilobit/second circuits, handle all network communication for their Hosts. To send a message to another Host, a Host precedes the text of its message with an address and simply delivers it to its IMP. The IMPs then determine the route, provide error control, and notify the sender of its receipt. The collection of Hosts, IMPs, and circuits forms the message switched resource sharing network. A good description of the ARPANET, and some early results on protocol development and modeling are given in References 3, 12, 15, 23 and 38. Some experimental utilization of the ARPANET is described in Reference 42. A more recent evaluation of such networks and a forward look is given in References 35 and 39.

The development of the Network involved four principal activities:

(1) The design of the IMPs to act as nodal store-and-forward switches,
(2) The topological design to specify the capacity and location of each communication circuit within the network,
(3) The design of higher level protocols for the use of the network by time-sharing, batch processing and other data processing systems, and
(4) System modeling and measurement of network performance.

Each of the first three activities were essentially performed independently of each other, whereas the modeling effort partly affected the IMP design effort, and closely interacted with the topological design project.

The IMPs were designed by Bolt Beranek and Newman Inc. (BBN) and were built to operate independent of the exact network connectivity; the topological structure was specified by Network Analysis Corporation (NAC) using models of network performance developed by NAC and by the University of California at Los Angeles (UCLA). The major efforts in the area of system modeling were performed at UCLA using theoretical and simulation techniques. Network performance measurements have been conducted during the development of the network by BBN and by the Network Measurement Center at UCLA. To facilitate effective use of the net, higher level (user) protocols are under development by a group of representatives of universities and research centers. This group, known as the Network Working Group, has already specified a Host to Host protocol and a Telnet protocol, and is in the process of completing other function oriented protocols.[4,29] We make no attempt to elaborate on the Host to Host protocol design problems in this paper.

THE NETWORK DESIGN PROBLEM

A variety of performance requirements and system constraints were considered in the design of the net. Unfortunately, many of the key design objectives had to be specified long before the actual user requirements could be known. Once the decision to employ message switching was made, and fifty kilobit/second circuits were chosen, the critical design variables were the network operating procedure and the network topology; the desired values of throughput, delay, reliability and cost were system performance and constraint variables. Other constraints affected the structure of the network, but not its overall properties, such as those arising from decisions about the length of time a message could remain within the network, the location of IMPs relative to location of Hosts, and the number of Hosts to be handled by a single IMP.

In this section, we identify the central issues related to IMP design, topological design, and network modeling. In the remainder of the paper, we describe the major design techniques which have evolved.

IMP properties

The key issue in the design of the IMPs was the definition of a relationship between the IMP subnet and the Hosts to partition responsibilities so that reliable and efficient operation would be achieved. The decision was made to build an autonomous subnet, independent (as much as possible) of the operation of any Host. The subnet was designed to function as a "communications system"; issues concerning the use of the subnet by the Hosts (such as protocol development) were initially left to the Hosts. For reliability, the IMPs were designed to be robust against all line failures and the vast majority of IMP and Host failures. This decision required routing strategies that dynamically adapt to changes in the states of IMPs and circuits,

and an elaborate flow control strategy to protect the subnet against Host malfunction and congestion due to IMP buffer limitations. In addition, a statistics and status reporting mechanism was needed to monitor the behavior of the network.

The number of circuits that an IMP must handle is a design constraint directly affecting both the structure of the IMP and the topological design. The speed of the IMP and the required storage for program and buffers depend directly upon the total required processing capacity, which must be high enough to switch traffic from one line to another when all are fully occupied. Of great importance is the property that all IMPs operate with identical programs. This technique greatly simplifies the problem of network planning and maintenance and makes network modifications easy to perform.

The detailed physical structure of the IMP is not discussed in this paper.[2,15] However, the operating procedure, which guides packets through the net, is very much of interest here. The flow control, routing, and error control techniques are integral parts of the operating procedure and can be studied apart from the hardware by which they are implemented. Most hardware modifications require changes to many IMPs already installed in the field, while a change in the operating procedure can often be made more conveniently by a change to the single operating program common to all IMPs, which can then be propagated from a single location via the net.

Topological properties

The topological design resulted in the specification of the location and capacity of all circuits in the network. Projected Host—Host traffic estimates were known at the start to be either unreliable or wrong. Therefore, the network was designed under the assumption of equal traffic between all pairs of nodes. (Additional superimposed traffic was sometimes included for those nodes with expectation of higher traffic requirements.) The topological structure was determined with the aid of specially developed heuristic programs to achieve a low cost, reliable network with a high throughput and a general insensitivity to the exact traffic distribution. Currently, only 50 kilobit/second circuits are being used in the ARPANET. This speed line was chosen to allow rapid transmission of short messages for interactive processing (e.g., less than 0.2 seconds average packet delay) as well as to achieve high throughput (e.g., at least 50 kilobits/second) for transmission of long messages. For reliability, the network was constrained to have at least two independent paths between each pair of IMPs.

The topological design problem requires consideration of the following two questions:

(1) Starting with a given state of the network topology, what circuit modifications are required to add or delete a set of IMPs?
(2) Starting with a given state of network topology, when and where should circuits be added or deleted to account for long term changes in network traffic?

If the locations of all network nodes are known in advance, it is clearly most efficient to design the topological structure as a single global effort. However, in the ARPANET, as in most actual networks, the initial designation of node locations is modified on numerous occasions. On each such occasion, the topology can be completely reoptimized to determine a new set of circuit locations.

In practice, there is a long lead time between the ordering and the delivery of a circuit, and major topological modifications cannot be made without substantial difficulty. It is therefore prudent to add or delete nodes with as little disturbance as possible to the basic network structure consistent with overall economical operation. Figure 1 shows the evolution of the ARPANET from the basic four IMP design in 1969 to the presently planned 27 IMP version. Inspection of the 24 and 27 IMP network designs reveals a few substantial changes in topology that take advantage of the new nodes being added. Surprisingly enough, a complete "reoptimization" of the 27 IMP topology yields a network only slightly less expensive (about 1 percent) than the present network design.[28]

Network models

The development of an accurate mathematical model for the evaluation of time delay in computer networks is among the more difficult of the topics discussed in this paper. On the one hand, the model must properly reflect the relevant features of the network structure and operation, including practical constraints. On the other hand, the model must result in a mathematical formulation which is tractable and from which meaningful results can be extracted. However, the two requirements are often incompatible and we search for an acceptable compromise between these two extremes.

The major modeling effort thus far has been the study of the behavior of networks of queues.[21] This emphasis is logical since in message switched systems, messages experience queueing delays as they pass from node to node and thus a significant performance measure is the

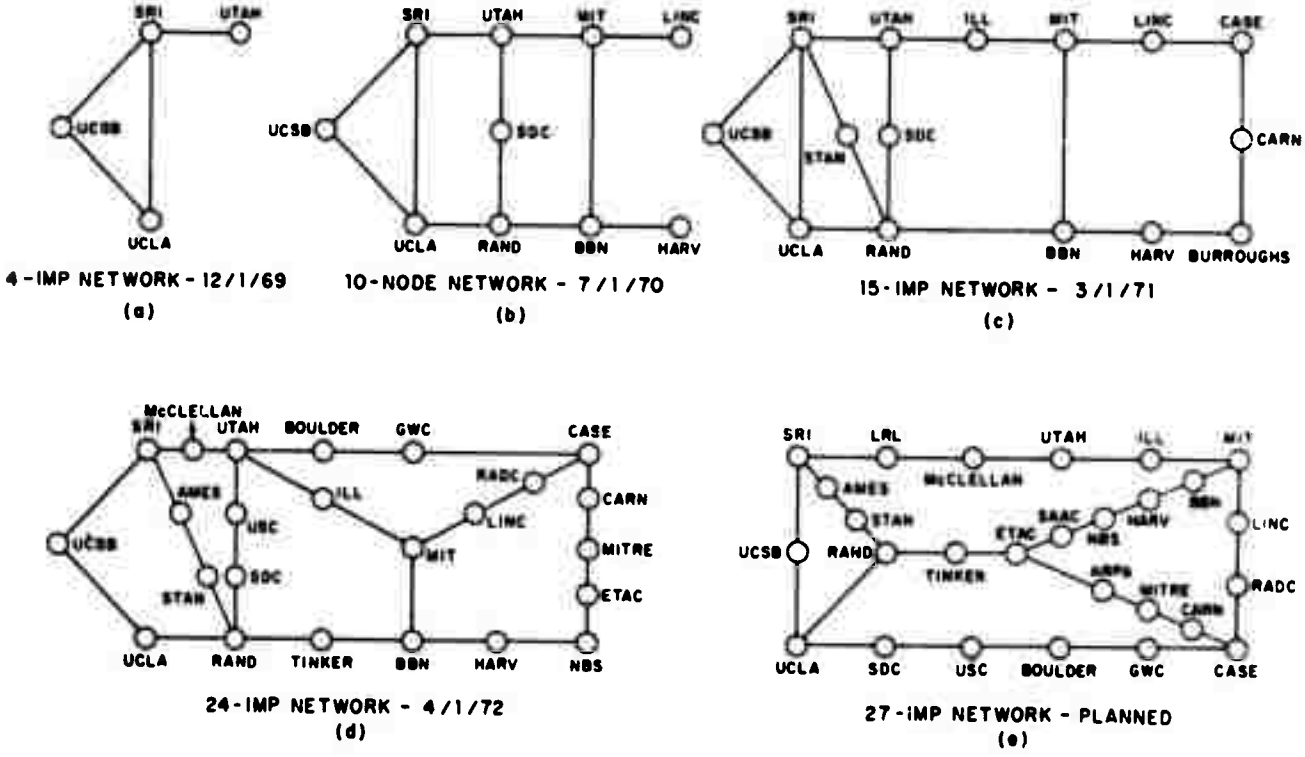


Figure 1—The evolution of the ARPANET

speed at which messages can be delivered. The queueing models were developed at a time when there were no operational networks available for experimentation and model validation, and simulation was the only tool capable of testing their validity. The models, which at all times were recognized to be idealized statements about the real network, were nonetheless crucial to the ARPANET topological design effort since they afforded the only known way to quantitatively predict the properties of different routing schemes and topological structures. The models have been subsequently demonstrated to be very accurate predictors of network throughput and indispensable in providing analytical insight into the network's behavior.

The key to the successful development of tractable models has been to factor the problem into a set of simpler queueing problems. There are also heuristic design procedures that one can use in this case. These procedures seem to work quite well and are described later in the paper. However, if one specializes the problem and removes some of the real constraints, theory and analysis become useful to provide understanding, intuition and design guidelines for the original constrained problem. This approach uncovers global properties of network behavior, which provide keys to good heuristic design procedures and ideal performance bounds.

DESIGN TECHNIQUES

In this section we describe the approaches taken to the design problems introduced in the previous section. We first summarize the important properties of the ARPANET design:

(1) A communications cost of less than 30 cents per thousand packets (approximately a megabit).
(2) Average packet delays under 0.2 seconds through the net.
(3) Capacity for expansion to 64 IMPs without major hardware or software redesign.
(4) Average total throughput capability of 10-15 kilobits/second for all Hosts at an IMP.
(5) Peak throughput capability of 85 kilobits/second per pair of IMPs in an otherwise unloaded network.
(6) Transparent communications with maximum message size of approximately 8000 bits and error rates of one bit in 10^{12} or less.

(7) Approximately 98 percent availability of any IMP and close to 100 percent availability of all operating IMPs from any operable IMP.

The relationships between the various design efforts are illustrated by these properties. The topological design provides for both a desired average throughput and for two or more paths to be fully used for communication between any pair of Hosts. The operating procedure should allow any pair of Hosts to achieve those objectives. The availability of IMPs to communicate reflects both the fact that IMPs are down about 2 percent of the time and that the topology is selected so that circuit failures contribute little additional to the total system downtime.

IMP design

The IMP design consists of two closely coupled but nonetheless separable pieces—the physical hardware specification (based on timing and reliability considerations and the operating procedure) and the design and implementation of the operating procedure using the specified IMP hardware. The IMP originally developed for the ARPANET contains a 16-bit one microsecond computer that can handle a total of about ¾ megabits/second of "useful" information on a total of approximately one megabit/second of circuit capacity (e.g., twenty 50 kilobit/second circuits). Hardware is likely to change as a function of the required IMP capacity but an operating procedure that operates well at one IMP capacity is likely to be transferable to machines that provide different capacity. However, as a network grows in size and utilization, a more comprehensive operating procedure that takes account of known structural properties, such as a hierarchical topology, is appropriate.

Four primary areas of IMP design, namely message handling and buffering, error control, flow control, and routing are discussed in this section. The IMP provides buffering to handle messages for its Host and packets for other IMPs. Error control is required to provide reliable communication of Host messages in the presence of noisy communication circuits. The design of the operating procedure should allow high throughput in the net under heavy traffic loads. Two potential obstacles to achieving this objective are: (1) The net can become congested and cause the throughput to *decrease* with increasing load, and (2) The routing procedure may be unable to always adapt sufficiently fast to the rapid movement of packets to insure efficient routing. A flow control and routing procedure is needed that can efficiently meet this requirement.

Message handling and buffering

In the ARPANET, the maximum message size was constrained to be approximately 8000 bits. A pair of Hosts will typically communicate over the net via a sequence of transmitted messages. To obtain delays of a few tenths of a second for such messages and to lower the required IMP buffer storage, the IMP program partitions each message into one or more packets each containing at most approximately 1000 bits. Each packet of a message is transmitted independently to the destination where the message is reassembled by the IMP before shipment to that destination Host. Alternately, the Hosts could assume the responsibility for reassembling messages. For an asynchronous IMP-Host channel, this marginally simplifies the IMP's task. However, if *every* IMP-Host channel were synchronous, and the Host provided the reassembly, the IMP task can be further simplified. In this latter case, "IMP-like" software would have to be provided in each Host.

The method of handling buffers should be simple to allow for fast processing and a small amount of program. The number of buffers should be sufficient to store enough packets for the circuits to be used to capacity; the size of the buffers may be intuitively selected with the aid of simple analytical techniques. For example, fixed buffer sizes are useful in the IMP for simplicity of design and speed of operation, but inefficient utilization can arise because of variable length packets. If each buffer contains A words of overhead and provides space for M words of text, and if message sizes are uniformly distributed between 1 and L, it can be shown[48] that the choice of M that minimizes the expected storage is approximately $\sqrt{AL}$. In practice, M is chosen to be somewhat smaller on the assumption that most traffic will be short and that the amount of overhead can be as much as, say, 25 percent of buffer storage.

Error control

The IMPs must assume the responsibility for providing error control. There are four possibilities to consider:

(1) Messages are delivered to their destination out of order.

(2) Duplicate messages are delivered to the destination.

(3) Messages are delivered with errors.

(4) Messages are not delivered.

The task of proper sequencing of messages for delivery to the destination Host actually falls in the province of both error control and flow control. If at most one message at a time is allowed in the net between a pair of Hosts, proper sequencing occurs naturally. A duplicate packet will arrive at the destination IMP after an acknowledgment has been missed, thus causing a successfully received packet to be retransmitted. The IMPs can handle the first two conditions by assigning a sequence number to each packet as it enters the network and processing the sequence number at the destination IMP. A Host that performs reassembly can also assign and process sequence numbers and check for duplicate packets. For many applications, the order of delivery to the destination is immaterial. For priority messages, however, it is typically the case that fast delivery requires a perturbation to the sequence.

Errors are primarily caused by noise on the communication circuits and are handled most simply by error detection and retransmission between each pair of IMPs along the transmission path. This technique requires extra storage in the IMP if either circuit speeds or circuit lengths substantially increase. Failures in detecting errors can be made to occur on the order of years to centuries apart with little extra overhead (20-30 parity bits per packet with the 50 kilobit/second circuits in the ARPANET). Standard cyclic error detection codes have been usefully applied here.

A reliable system design insures that each transmitted message is accurately delivered to its intended destination. The occasional time when an IMP fails and destroys a useful in-transit message is likely to occur far less often than a similar failure in the Hosts and has proven to be unimportant in practice, as are errors due to IMP memory failures. A simple end to end retransmission strategy will protect against these situations, if the practical need should arise. However, the IMPs are designed so that they can be removed from the network without destroying their internally stored packets.

Flow control

A network in which packets may freely enter and leave can become congested or logically deadlocked and cause the movement of traffic to halt.[8,17] Flow control techniques are required to prevent these conditions from occurring. The provision of extra buffer storage will mitigate against congestion and deadlocks, but cannot in general prevent them.

The sustained failure of a destination Host to accept packets from its IMP at the rate of arrival will cause the net to fill up and become congested. Two kinds of logical deadlocks, known as reassembly lockup and store-and-forward lockup may also occur. In reassembly lockup, the remaining packets of partially reassembled messages are blocked from reaching the destination IMP (thus preventing the message from being completed and the reassembly space freed) by other packets in the net that are waiting for reassembly space at that destination to become free. In a store-and-forward lockup, the destination has room to accept arriving packets, but the packets interfere with each other by tying up buffers in transit in such a way that none of the packets are able to reach the destination.[17] These phenomena have only been made to occur during very carefully arranged testing of the ARPANET and by simulation.[19]

In the original ARPANET design, the use of software links and RFNMS protected against congestion by a single link or a small set of links. However, the combined traffic on a large number of links could still produce congestion. Although this strategy did not protect against lockup, the method has provided ample protection for the levels of traffic encountered by the net to date.

A particularly simple flow control algorithm that augments the original IMP design to prevent congestion and lockup is also described in Reference 17. This scheme includes a mechanism whereby packets may be discarded from the net at the destination IMP when congestion is about to occur, with a copy of each discarded packet to be retransmitted a short time later by the originating Host's IMP. Rather than experience excessive delays within the net as traffic levels are increased, the traffic is queued outside the net so that the transit time delays internal to the net continue to remain small. This strategy prevents the insertion of more traffic into the net than it can handle.

It is important to note the dual requirement for small delays for interactive traffic and high bandwidth for the fast transfer of files. To allow high bandwidth between a pair of Hosts, the net must be able to accept a steady flow of packets from one Host and at the same time be able to rapidly quench the flow at the entrance to the source IMP in the event of imminent congestion at the destination. This usually requires that a separate provision be made in the algorithm to protect short interactive messages from experiencing unnecessarily high delays.

Routing

Network routing strategies for distributed networks require routing decisions to be made with only information available to an IMP and the IMP must

execute those decisions to effect the routing.[14,15] A simple example of such a strategy is to have each IMP handling a packet independently route it along its current estimate of the shortest path to the destination.

For many applications, it suffices to deal with an idealized routing strategy which may not simulate the IMP routing functions in detail or which uses information not available to the IMP. The general properties of both strategies are usually similar, differing mainly in certain implementation details such as the availability of buffers or the constraint of counters and the need for the routing to quickly adapt to changes in IMP and circuit status.

The IMPs perform the routing computations using information received from other IMPs and local information such as the alive/dead state of its circuits. In the normal case of time varying loads, local information alone, such as the length of internal queues, is insufficient to provide an efficient routing strategy without assistance from the neighboring IMPs. It is possible to obtain sufficient information from the neighbors to provide efficient routing, with a small amount of computation needed per IMP and without each IMP requiring a topological map of the network. In certain applications where traffic patterns exhibit regularity, the use of a central controller might be preferable. However, for most applications which involve dynamically varying traffic flow, it appears that a central controller cannot be used more effectively than the IMPs to update routing tables if such a controller is constrained to derive its information via the net. It is also a less reliable approach to routing than to distribute the routing decisions among the IMPs.

The routing information cannot be propagated about the net in sufficient time to accurately characterize the instantaneous traffic flow. An efficient algorithm, therefore, should not focus on the movement of individual packets, but rather use topological or statistical information in the selection of routes. For example, by using an averaging procedure, the flow of traffic can be made to build up smoothly. This allows the routing algorithm ample time to adjust its tables in each IMP in advance of the build-up of traffic.

The scheme originally used in the ARPA network had each IMP select one output line per destination onto which to route packets. The line was chosen to be the one with minimum estimated time delay to the destination. The selection was updated every half second using minimum time estimates from the neighboring IMPs and internal estimates of the delay to each of the neighbors. Even though the routing algorithm only selects one line at a time per destination, two output lines will be used if a queue of packets waiting transmission on one line builds up before the routing update occurs and another line is chosen. Modifications to the scheme which allow several lines per destination to be used in an update interval (during which the routing is not changed) are possible using two or more time delay estimates to select the paths.

In practice, this approach has worked quite effectively with the moderate levels of traffic experienced in the net. For heavy traffic flow, this strategy may be inefficient, since the routing information is based on the length of queues, which we have seen can change much faster than the information about the change can be distributed. Fortunately, this information is still usable, although it can be substantially out of date and will not, in general, be helpful in making efficient routing decisions in the heavy traffic case.

A more intricate scheme, recently developed by BBN, allows multiple paths to be efficiently used even during heavy traffic.[18] Preliminary simulation studies indicate that it can be tailored to provide efficient routing in a network with a variety of heavy traffic conditions. This method separates the problem of defining routes onto which packets may be routed from the problem of selecting a route when a particular packet must be routed. By this technique, it is possible to send packets down a path with the fewest IMPs and excess capacity, or when that path is filled, the one with the next fewest IMPs and excess capacity, etc.

A similar approach to routing was independently derived by NAC using an idealized method that did not require the IMPs to participate in the routing decisions. Another approach using a flow deviation technique has recently been under study at UCLA.[12] The intricacies of the exact approach lead to a metering procedure that allows the overall network flow to be changed slowly for stability and to perturb existing flow patterns to obtain an increased flow. These approaches all possess, in common, essential ingredients of a desirable routing strategy.

Topological considerations

An efficient topological design provides a high throughput for a given cost. Although many measures of throughput are possible, a convenient one is the average amount of traffic that a single IMP can send into the network when all other IMPs are transmitting according to a specified traffic pattern. Often, it is assumed that all other IMPs are behaving identically and each IMP is sending equal amounts of traffic to each other IMP. The constraints on the topological design are the available common carrier circuits, the target cost or throughput, the desired reliability, and

TABLE I—23 Node 28 Link ARPA

Number of Circuits Failed	Number of Combinations to be Examined	Number of Cutsets
28	1	1
27	28	28
26	378	378
25	3276	3276
24	20475	20475
23	98280	98280
22	376740	376740
21	1184040	1184040
20	3108105	3108105
19	6906900	6906900
18	13123110	13123110
17	21474180	21474180
16	30421755	30421755
15	37442160	37442160
14	40116600	40116600
13	37442160	37442160
12	30421755	30421755
11	21474180	21474180
10	13123110	13123110
9	6906900	6906900
8	3108108	3108108
7	1184040	1184040
6	376740	349618
5	98280	≈70547
4	20475	≈9852
3	3276	827
2	378	30
1	28	0

the cost of computation required to perform the topological design.

Since, there was no clear specification of the amount of traffic that the network would have to accommodate initially, it was first constructed with enough excess capacity to accommodate any reasonable traffic requirements. Then as new IMPs were added to the system, the capacity was and is still being systematically reduced until the traffic level occupies a substantial fraction of the network's total capacity. At this point, the net's capacity will be increased to maintain the desired percentage of loading. At the initial stages of network design, the "two-connected" reliability constraint essentially determined a minimum value of maximum throughput. This constraint forces the average throughput to be in the range 10-15 kilobits per second per IMP, when 50 kilobit/sec circuits are used throughout the network, since two communication paths between every pair of IMPs are needed. Alternatively, if this level of throughput is required, then the reliability specification of "two-connectivity" can be obtained without additional cost.

Reliability computations

A simple and natural characterization of network reliability is the ability of the network to sustain communication between all operable pairs of IMPs. For design purposes, the requirement of two independent paths between nodes insures that at least two IMPs and/or circuits must fail before any pair of operable IMPs cannot communicate. This criterion is independent of the properties of the IMPs and circuits, does not take into account the "degree" of disruption that may occur and hence, does not reflect the actual availability of resources in the network. A more meaningful measure is the average fraction of IMP pairs that cannot communicate because of IMP and circuit failures. This calculation requires knowledge of the IMP and circuit failure rates, and could not be performed until enough operating data was gathered to make valid predictions.

To calculate network reliability, we must consider elementary network structures known as cutsets. A

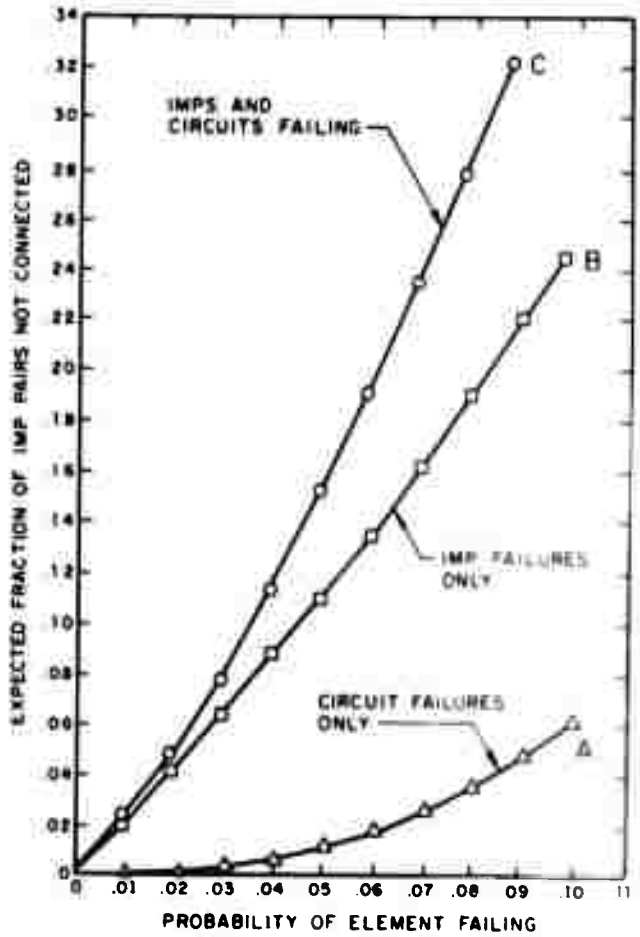


Figure 2—Network availability vs. IMP and circuit reliability

cutset is a set of circuits and or IMPs whose removal from the network breaks all communication paths between at least two operable IMPs. To calculate reliability, it is often the case that all cutsets must be either enumerated or estimated. As an example, in a 23 IMP, 28 circuit ARPA Network design similar to the one shown in Figure 1(d), there are over twenty million ways of deleting only circuits so that the remaining network has at least one operable pair of IMPs with no intact communication paths. Table 1 indicates the numbers of cutsets in the 23 IMP network as a function of the number of circuits they contain.

A combination of analysis and simulation can be used to compute the average fraction of non-communicating IMP pairs. Detailed descriptions of the analysis methods are given in Reference 44 while their application to the analysis of the ARPANET is discussed in Reference 43. The results of an analysis of the 23 IMP version of the network are shown in Figure 2. The curve marked A shows the results under the assumption that IMPs do not fail, while the curve marked B shows the case where circuits do not fail. The curve marked C assumes that both IMPs and circuits fail with equal probability. In actual operation, the average failure probability of both IMPs and circuits is about 0.02. For this value, it can be seen that the effect of circuit failures is far less significant than the effect of IMP failures. If an IMP fails in a network with n IMPs, at least $n-1$ other IMPs cannot communicate with it. Thus, good network design cannot improve upon the effect directly due to IMP failures, which in the ARPANET is the major factor affecting the reliability of the communications. Further, more intricate reliability analyses which consider the loss of throughput capacity because of circuit failures have also been performed and these losses have been shown to be negligible.[28] Finally, unequal failure rates due to differences in line lengths have been shown to have only minor effects on the analysis and can usually be neglected.[27]

Topological optimization

During the computer optimization process, the reliability of the topology is assumed to be acceptable if the network is at least two-connected. The object of the optimization is to decrease the ratio of cost to throughput subject to an overall cost limitation. This technique employs a sophisticated network optimization program that utilizes circuit exchange heuristics, routing and flow analysis algorithms, to generate low cost designs. In addition, two time delay models were initially used to (1) calculate the throughput corresponding to an average time delay of 0.2 seconds, (2) estimate the packet rejection rate due to all buffers filling at an IMP. As experience with these models grew, the packet rejection rate was found to be negligible and the computation discontinued. The delay computation (Equation (7) in a later section) was subsequently first replaced by a heuristic calculation to speed the computation and later eliminated after it was found that time delays could be guaranteed to be acceptably low by preventing cutsets from being saturated. This "threshold" behavior is discussed further in the next section.

The basic method of optimization was described in Reference 12 while extensions to the design of large networks are discussed in Reference 9. The method operates by initially generating, either manually or by computer, a "starting network" that satisfies the overall network constraints but is not, in general, a low cost network. The computer then iteratively modifies the starting network in simple steps until a lower cost network is found that satisfies the constraints or the process is terminated. The process is repeated until no further improvements can be found. Using a different starting network can result in a different solution. However, by incorporating sensible heuristics and by using a variety of *carefully chosen* starting networks and some degree of man-machine interaction, "excellent" final networks usually result. Experience has shown that there are a wide variety of such networks with different topological structures but similar cost and performance.

The key to this design effort is the heuristic procedure by which the iterative network modifications are made. The method used in the ARPANET design involves the removal and addition of one or two circuits at a time. Many methods have been employed, at various times, to identify the appropriate circuits for potential addition or deletion. For example, to delete uneconomical circuits a straightforward procedure simply deletes single circuits in numerical order, reroutes traffic and reevaluates cost until a decrease in cost per megabit is found. At this point, the deletion is made permanent and the process begins again. A somewhat more sophisticated procedure deletes circuits in order of increasing utilization, while a more complex method attempts to evaluate the effect of the removal of any circuit before any deletion is attempted. The circuit with the greatest likelihood of an improvement is then considered for removal and so on.

There are a huge number of reasonable heuristics for circuit exchanges. After a great deal of experimentation with many of these, it appears that the choice of a particular heuristic is not critical. Instead, the speed and efficiency with which potential exchanges can be

investigated appears to be the limiting factor affecting the quality of the final design. Finally, as the size of the network increases, the greater the cost becomes to perform *any* circuit exchange optimization. Decomposition of the network design into regions becomes necessary and additional heuristics are needed to determine effective decompositions. It presently appears that these methods can be used to design relatively efficient networks with a few hundred IMPs while substantially new procedures will be necessary for networks of greater size.

The topological design requires a routing algorithm to evaluate the throughput capability of any given network. Its properties must reflect those of an implementable routing algorithm, for example, within the ARPANET. Although the routing problem can be formulated as a "multicommodity flow problem"[10] and solved by linear programming for networks with 20-30 IMPs,[8] faster techniques are needed when the routing algorithm is incorporated in a design procedure. The design procedure for the ARPA Network topology iteratively analyzes thousands of networks. To satisfy the requirements for speed, an algorithm which selects the least utilized path with the minimum number of IMPs was initially used.[12] This algorithm was later replaced by one which sends as much traffic as possible along such paths until one or more circuits approach a few percent of full utilization.[28] These highly utilized circuits are then no longer allowed to carry additional flow. Instead, new paths with excess capacity and possibly more intermediate nodes are found. The procedure continues until some cutset contains only nearly fully utilized circuits. At this point no additional flow can be sent. For design purposes, this algorithm is a highly satisfactory replacement for the more complicated multi-commodity approach. Using the algorithm, it has been shown that the throughput capabilities of the ARPA Network are substantially insensitive to the distribution of traffic and depend mainly only on the total traffic flow within the network.[8]

Analytic models of network performance

The effort to determine analytic models of system performance has proceeded in two phases: (1) the prediction of average time delay encountered by a message as it passes through the network, and (2) the use of these queueing models to calculate optimum channel capacity assignments for minimum possible delay. The model used as a standard for the average message delay was first described in Reference 21 where it served to predict delays in stochastic communication networks. In Reference 22, it was modified to describe the behavior of ARPA-like computer networks while in Reference 23 it was refined further to apply directly to the ARPANET.

The single server model

Queueing theory[20] provides an effective set of analytical tools for studying packet delay. Much of this theory considers systems in which messages place demands for transmission (service) upon a single communication channel (the single server). These systems are characterized by $A(\tau)$, the distribution of interarrival times between demands and $B(t)$, the distribution of service times. When the average demand for service is less than the capacity of the channel, the system is said to be stable.

When $A(\tau)$ is exponential (i.e., Poisson arrivals), and messages are transmitted on a first-come first-served basis, the average time T in the stable system is

$$T = \frac{\lambda \bar{t}^2}{2(1-\rho)} + \bar{t} \qquad (1)$$

where λ is the average arrival rate of messages, $\bar{t}$ and $\bar{t}^2$ are the first and second moments of $B(t)$ respectively, and $\rho = \lambda \bar{t} < 1$. If the service time is also exponential,

$$T = \frac{\bar{t}}{1-\rho} \qquad (2)$$

When $A(\tau)$ and $B(t)$ are arbitrary distributions, the situation becomes complex and only weak results are available. For example, no expression is available for T; however the following upper bound yields an excellent approximation[19] as $\rho \rightarrow 1$:

$$T \leq \frac{\lambda(\sigma_a^2 + \sigma_b^2)}{2(1-\rho)} + \bar{t} \qquad (3)$$

where σ_a^2 and σ_b^2 are the variance of the interarrival time and service time distributions, respectively.

Networks of queues

Multiple channels in a network environment give rise to queueing problems that are far more difficult to solve than single server systems. For example, the variability in the choice of source and destination for a message is a network phenomenon which contributes to delay. A principal analytical difficulty results from the fact that flows throughout the network are correlated. The basic approach to solving these stochastic network

problems is to *decompose* them into analyzable single-server problems which reflect the original network structure and traffic flow.

Early studies of queueing networks indicated that such a decomposition was possible;[50,51] however, those results do not carry over to message switched computer networks due to the correlation of traffic flows. In Reference 21 it was shown for a wide variety of communication nets that this correlation could be removed by considering the length of a given packet to be an independent random variable as it passes from node to node. Although this "independence" assumption is not physically realistic, it results in a mathematically tractable model which does not seem to affect the accuracy of the predicted time delays. As the size and connectivity of the network increases, the assumption becomes increasingly more realistic. With this assumption, a successful decomposition which permits a channel-by-channel analysis is possible, as follows.

The packet delay is defined as the time which a packet spends in the network from its entry until it reaches its destination. The average packet delay is denoted as T. Let Z_{jk} be the average delay for those packets whose origin is IMP j and whose destination is IMP k. We assume a Poisson arrival process for such packets with an average of γ_{jk} packets per second and an exponential distribution of packet lengths with an average of $1/\mu$ bits per packet. With these definitions, if γ is the sum of the quantities γ_{jk}, then[21]

$$T = \sum_{j,k} \frac{\gamma_{jk}}{\gamma} Z_{jk} \tag{4}$$

Let us now reformulate Equation (4) in terms of single channel delays. We first define the following quantities for the ith channel: C_i as its capacity (bits/second); λ_i as the average packet traffic it carries (packets/second); and T_i as the average time a packet spends waiting for and using the ith channel. By relating the $\{\lambda_i\}$ to the $\{\gamma_{jk}\}$ via the paths selected by the routing algorithm, it is easy to see that[21]

$$T = \sum_i \frac{\lambda_i}{\gamma} T_i \tag{5}$$

With the assumption of Poisson traffic and exponential service times, the quantities T_i are given by Equation (2). For an average packet length of $1/\mu$ bits, $\bar{t} = 1/\mu C_i$ seconds and thus

$$T_i = \frac{1}{\mu C_i - \lambda_i} \tag{6}$$

Thus we have successfully decomposed the analysis problem into a set of simple single-channel problems.

A refinement of the decomposition permits a non-exponential packet length distribution and uses Equation (1) rather than Equation (2) to calculate T_i; as an approximation, the Markovian character of the traffic is assumed to be preserved. Furthermore, for computer networks we include the effect of propagation time and overhead traffic to obtain the following equation for average packet delay[22,23]

$$T = K + \sum_i \frac{\lambda_i}{\gamma}\left[\frac{1}{\mu' C_i} + \frac{\lambda_i/\mu C_i}{\mu C_i - \lambda_i} + P_i + K\right] \tag{7}$$

Here, $1/\mu'$ represents the average length of a Host packet, and $1/\mu$ represents the average length of all packets (including acknowledgments, headers, requests for next messages, parity checks, etc.) within the network. The expression $1/\mu' C_i + [(\lambda_i/\mu C_i)/(\mu C_i - \lambda_i)] + P_i$ represents the average packet delay on the ith channel. The term $(\lambda_i/\mu C_i)/(\mu C_i - \lambda_i)$ is the average time a packet spends waiting at the IMP for the ith channel to become available. Since the packet must compete with acknowledgments and other overhead traffic, the overall average packet length $1/\mu$ appears in the expression. The term $1/\mu' C_i$ is the time required to transmit a packet of average length μ'. Finally: K is the nodal processing time, assumed constant, and for the ARPA IMP approximately equal to 0.35 ms; P_i is the propagation time on the ith channel (about 20 ms for a 3000 mile channel).

Assuming a relatively homogeneous set of C_i and P_i, no individual term in the expression for delay will dominate the summation until the flow λ_i/μ in one channel (say channel i_0) approaches the capacity C_{i_0}. At that point, the term T_{i_0}, and hence T will grow rapidly. The expression for delay is then dominated by one (or more) terms and exhibits a threshold behavior. Prior to this threshold, T remains relatively constant.

The accuracy of the time delay model, as well as this threshold phenomenon was demonstrated on a 19 node network[14] and on the ten node ARPA net derived from Figure 1(c) by deleting the rightmost five IMPs. Using the routing procedure described in the last section[28] and equal traffic between all node pairs, the channel flows λ_i were found for the ten node net and the delay curves shown in Figure 3 were obtained. Curve A was obtained with fixed 1000 bit packets,* while curve B was generated for exponentially distributed variable length packets with average size of 500 bits. In both cases A and B, all overhead factors were ignored. Note that the delay remains small until a

* In case A, the application of Equation (1) allows for constant packet lengths (i.e., zero variance).

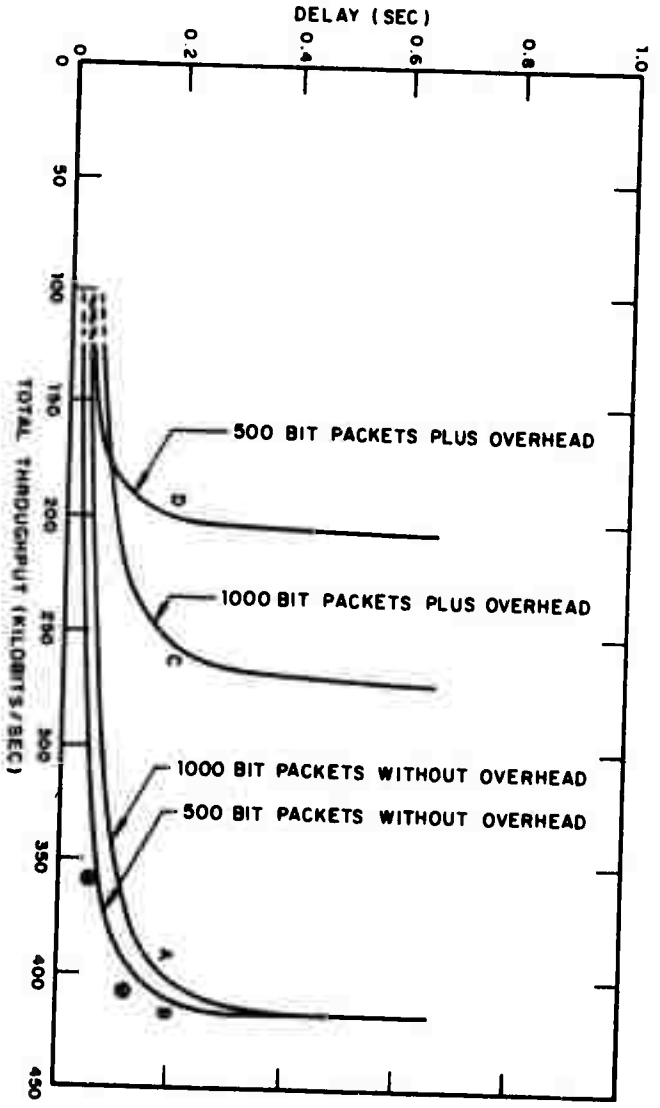


Figure 3—Delay vs. throughput

total throughput slightly greater than 400 kilobits/second is reached. The delay then increases rapidly. Curves C and D respectively represent the same situations where the overhead of 136 bits per packet and per RFNM and 152 bits per acknowledgment are included. Notice that the total throughput per IMP is reduced to 250 kilobits/second in case C and to approximately 200 kilobits/second in case D.

In the same figure, we have illustrated with x's the results of a simulation performed with a realistic routing and metering strategy. The simulation omitted all network overhead and assumed fixed lengths of 1000 bits for all packets.

It is difficult to develop a practical routing and flow control procedure that will allow each IMP to input identical amounts of traffic. To compare the delay curve A with the points obtained by simulation, the curve should actually be recomputed for the slightly skewed distribution that resulted. It is notable that the delay estimates from the simulation (which used a dynamic routing strategy) and the computation (which used a static routing strategy and the time delay formula) are in close agreement. In particular, they both accurately determined the vertical rise of the delay curve in the range just above 400 kilobits/second, the formula by predicting infinite delay and the simulation by rejecting the further input of traffic.

In practice and from the analytic and simulation studies of the ARPANET, the average queueing delay is observed to remain small (almost that of an unloaded net) and well within the design constraint of 0.2 seconds until the traffic within the network approaches the capacity of a cutset. The delay then increases rapidly. *Thus, as long as traffic is low enough and the routing adaptive enough to avoid the premature saturation of cutsets by guiding traffic along paths with excess capacity, queueing delays are not significant.*

Channel capacity optimization

One of the most difficult design problems is the optimal selection of capacities from a *finite* set of options. Although there are many heuristic approaches to this problem, analytic results are relatively scarce. (For the specialized case of centralized networks, an algorithm yielding optimal results is available.[11]) While it is possible to find an economical assignment of discrete capacities for, say, a 200 IMP network, very little is known about the relation between such capacity assignments, message delay, and cost.

To obtain theoretical properties of optimal capacity assignments, one may ignore the constraint that capacities are obtainable only in discrete sizes. In Reference 21 such a problem was posed where the network topology and average traffic flow were assumed to be known and fixed and an optimal match of capacities to traffic flow was found. Also, the traffic was assumed to be Markovian (Poisson arrivals and exponential packet lengths) and the independence assumption and decomposition method were applied. For each channel, the capacity C_i was found which minimized the average message delay T, at a fixed total system cost D. Since λ_i/μ is the average bit rate on the ith channel, the solution to *any* optimal assignment problem must provide more than this minimal capacity to each channel. This is clear since both Equations (6) and (7) indicate that T_i will become arbitrarily large with less than (or equal to) this amount of capacity. It is not critical exactly how the *excess* capacity is

assigned, as long as $C_i > \lambda_i/\mu$. Other important parameters and insights have been identified in studying the continuous capacity optimization problem. For example, the number of excess dollars, D_e, remaining after the minimum capacity λ_i/μ is assigned to each channel is of great importance. As $D_e \to 0$, the average delay must grow arbitrarily large. In this range, the critical parameters become ρ and $\bar{n}$ where $\rho = \gamma/\mu C$ is the ratio of the rate γ/μ at which bits enter the network to the rate C at which the net can handle bits and $\bar{n} = \lambda/\gamma$, where $\lambda = \sum \lambda_i$ is the total rate at which packets flow within the net. The quantity ρ represents a dimensionless form of network "load" whereas $\bar{n}$ is easily shown to represent the average path length for a packet.

As the load ρ approaches $1/\bar{n}$, the delay T grows very quickly, and this point $\rho = 1/\bar{n}$ represents the maximum load which the network can support. If capacities are assigned optimally, all channels saturate simultaneously at this point. In this formulation $\bar{n}$ is a design parameter which depends upon the topology and the routing procedure, while ρ is a parameter which depends upon the input rate and the total capacity of the network. In studying the ARPANET[23] a closer representation of the actual tariffs for high speed telephone data channels used in that network was provided by setting $D = \sum_i d_i C_i$ where $0 \leq \alpha \leq 1$.* This approach requires the solution of a non-linear equation by numerical techniques. On solving the equation, it can be shown that the packet delay T varies insignificantly with α for $.3 \leq \alpha \leq 1$. This indicates that the closed form solution discussed earlier with $\alpha = 1$ is a reasonable approximation to the more difficult non-linear problem. These continuous capacity studies have application to general network studies (e.g., satellite communications)[33] and are under continued investigation.[25,26,46]

In practice, the selection of channel capacities must be made from a small finite set. Although some theoretical work has been done in this case by approximating the discrete cost-capacity functions by continuous ones, much remains to be done.[13,25] Because of the discrete capacities and the time varying nature of network traffic, it is *not* generally possible to match channel capacities to the anticipated flows within the channels. If this were possible, all channels would saturate at the same externally applied load. Instead, capacities are assigned on the basis of reasonable estimates of average or peak traffic flows. It is the responsibility of the routing procedure to permit the traffic to adapt to the available capacity.[14] Often two IMP sites will engage in heavy communication and thus saturate one or more critical network cutsets. In such cases, the routing will not be able to send additional flow across these cuts. The network will therefore experience "premature" saturation in one or a small set of channels leading to the threshold behavior described earlier.

DISCUSSION

A major conclusion from our experience in network design is that message switched networks of the ARPA type are no longer difficult to specify. They may be implemented straightforwardly from the specifications; they can be less expensive than other currently available technical approaches; they perform remarkably well as a communication system for interconnecting time-sharing and batch processing computers and can be adapted to directly handle teletypes, displays and many other kinds of terminal devices and data processing equipment.[16,30]

The principal tools available for the design of networks are analysis, simulation, heuristic procedures, and experimentation. Analysis, simulation and heuristics have been the mainstays of the work on modeling and topological optimization while simulation, heuristic procedures and experimental techniques have been the major tools for the actual network implementation. Experience has shown that all of these methods are useful while none are all powerful. The most valuable approach has been the simultaneous use of several of these tools.

Each approach has room for considerable improvement. The analysis efforts have not yet yielded results in many important areas such as routing. However, for prediction of delay, this approach leads to a simple threshold model which is both accurate and understandable. Heuristic procedures all suffer from the problem that it is presently unclear how to select appropriate heuristics. It has been the innovative use of computers and analysis that has made the approach work well. For designing networks with no more than a few hundred IMPs, present heuristics appear adequate but a good deal of additional work is required for networks of greater size. Simulation is a well developed tool that is both expensive to apply and limited in the overall understanding that it yields. For these reasons, simulation appears to be most useful only in validating models, and in assisting in detailed design decisions such as the number of buffers that an IMP should contain. As the size of networks continues to grow, it appears that simulation will become virtually useless as a total design tool. The ultimate standard by which all models and

* Of course the tariffs reflect the discrete nature of available channels. The use of the exponent α provides a continuous fit to the discrete cost function. For the ARPANET, $\alpha \cong .8$.

conclusions can be tested is experimentation. Experimentation with the actual network is conceptually relatively straightforward and very useful. Although, experiments are often logistically difficult to perform, they can provide an easy means for testing models, heuristics and design parameters.

The outstanding design problems currently facing the network designer are to specify and determine the properties of the routing, flow control and topological structure for large networks. This specification must make full use of a wide variety of circuit options. Preliminary studies indicate that initially, the most fruitful approaches will be based on the partitioning of the network into regions, or equivalently, constructing a large network by connecting a number of regional networks. To send a message, a Host would specify both the destination region and the destination IMP in that region. No detailed implementation of a large network has yet been specified but early studies of their properties indicate that factors such as cost, throughput, delay and reliability are similar to those of the present ARPANET, if the ARPA technology is used.[9]

Techniques applicable to the design of large networks are presently under intensive study. These techniques appear to split into the same four categories as small network design but approaches may differ significantly. For example, large nets are likely to demand the placement of high bandwidth circuits at certain key locations in the topology to concentrate flow. These circuits will require the development of a high speed IMP to connect them into the net. It is likely that this high speed IMP will have the structure of a high speed multiplexor, and may require several cooperating processors to obtain the needed computer power for the job. Flow control strategies for large networks seem to extrapolate nicely from small network strategies if each region in the large network is viewed as a node in a smaller network. However, this area will require additional study as will the problem of specifying effective adaptive routing mechanisms. Recent efforts indicate that efficient practical schemes for small networks will soon be available. These schemes seem to be applicable for adaptive routing and flow control in networks constructed from regional subnetworks. The development of practical algorithms to handle routing and flow control is still an art rather than a science. Simulation is useful for studying the properties of a given heuristic, but intuition still plays a dominant role in the system design.

Several open questions in network design presently are: (1) what structure should a high bandwidth IMP have; (2) how can full use be made of a variety of high bandwidth circuits; (3) how should large networks be partitioned for both effective design and operation; and (4) what operational procedures should large networks follow? Much work has already been done in these areas but much more remains to be done. We expect substantial progress to be achieved in the next few years and accordingly, the interest and understanding of the properties of message switched networks of all sizes.

ACKNOWLEDGMENT

The ARPA Network is in large part the conception of Dr. L. G. Roberts of the Advanced Research Projects Agency to whom we owe a debt of gratitude for his support and encouragement. We also acknowledge the helpful contributions of S. Crocker and B. Dolan of ARPA. At BBN, NAC, and UCLA many individuals, too numerous to list, participated in the network effort and we gratefully acknowledge their contributions.

REFERENCES

1 P BARAN S BOEHM P SMITH
On distributed communications
Series of 11 reports by Rand Corporation Santa Monica California 1964

2 *"Specifications for the interconnection of a Host and an IMP*
BBN Report No 1822 1971 revision

3 S CARR S CROCKER V CERF
Host-Host communication protocol in the ARPA network
SJCC 1970 pp 589-597

4 S CROCKER et al
Function oriented protocols for the ARPA network
SJCC 1972 in this issue

5 D W DAVIES
The control of congestion in packet switching networks
Proc of the Second ACM IEEE Symposium on problems in the Optimization of Data Communications Systems Palo Alto California Oct 1971 pp 46-49

6 D FARBER K LARSON
The architecture of a distributed computer system—An informal description
University of California Irvine Information and Computer Science Technical Report #11 1970

7 W D FARMER E E NEWHALL
An experimental distribution switching system to handle bursty computer traffic
Proc of the ACM Symposium on Problems in the Optimization of Data Communication Systems 1969 pp 1-34

8 H FRANK W CHOU
Routing in computer networks
Networks John Wiley 1971 Vol 1 No 2 pp 99-112

9 H FRANK W CHOU
Cost and throughput in computer-communication networks
To appear in the Infotech Report on the State of the Art of Computer Networks 1972

10 H FRANK I T FRISCH
Communication transmission and transportation networks
Addison Wesley 1972

11 H FRANK I T FRISCH W CHOU
R VAN SLYKE
Optimal design of centralized computer networks
Networks John Wiley Vol 1 No 1 pp 43-57 1971
12 H FRANK I T FRISCH W CHOU
Topological considerations in the design of the ARPA computer network
SJCC May 1970 pp 581-587
13 L FRATTA M GERLA L KLEINROCK
The flow deviation method; An approach to store-and-forward network design
To be published
14 G FULTZ L KLEINROCK
Adaptive routing techniques for store-and-forward computer-communication networks
Proc of the International Conference on communications 1971 pp 39-1 to 39-8
15 F HEART R KAHN S ORNSTEIN
W CROWTHER D WALDEN
The interface message processor for the ARPA computer network
SJCC 1970 pp 551-567
16 R E KAHN
Terminal access to the ARPA computer network
Proc of Third Courant Institute Symposium Nov 1970
To be published by Prentice Hall Englewood Cliffs, NJ
17 R E KAHN W R CROWTHER
Flow control in a resource sharing computer network
Proc of the Second ACM IEEE Symposium on Problems in the Optimization of Data Communications Systems Palo Alto California October 1971 pp 108-116
18 R E KAHN W R CROWTHER
A study of the ARPA network design and performance
BBN Report No 2161 August 1971
19 J F C KINGMAN
Some inequalities for the queue GI/G/1
Biometrica 1962 pp 315-324
20 L KLEINROCK
Queueing systems; Theory and application
To be published by John Wiley 1972
21 L KLEINROCK
Communication nets: Stochastic message flow and delay
McGraw-Hill 1964
22 L KLEINROCK
Models for computer networks
Proc of the International Conference on Communications 1969 pp 21-9 to 21-16
23 L KLEINROCK
Analysis and simulation methods in computer network design
SJCC 1970 pp 569-579
24 T MARILL L G ROBERTS
Toward a cooperative network of time-shared computers
FJCC 1966
25 B MEISTER H MULLER H RUDIN JR
Optimization of a new model for message-switching networks
Proc of the International Conference on Communications 1971 pp 39-16 to 39-21
26 B MEISTER H MULLER H RUDIN
New optimization criteria for message-switching networks
IEEE Transactions on Communication Technology Com-19 June 1971 pp 256-260
27 N. A. C. Third Semiannual Technical Report for the Project Analysis and Optimization of Store-and-Forward Computer Networks
Defense Documentation Center Alexandria Va June 1971
28 N. A. C. Fourth Semiannual Technical Report for the Project Analysis and Optimization of Store-and-Forward Computer Networks
Defense Documentation Center Alexandria Va Dec 1971
29 *The Host/Host protocol for the ARPA network*
Network Working Group N I C No 7147 1971 (Available from the Network Information Center Stanford Research Institute Menlo Park California)
30 ORNSTEIN et al
The terminal IMP for the ARPA network
SJCC 1972 In this issue
31 J PIERCE
A network for block switching of data
IEEE Convention Record New York N Y March 1971
32 E PORT F CLOS
Comparisons of switched data networks on the basis of waiting times
IBM Research Report RZ 405 IBM Research Laboratories Zurich Switzerland Jan 1971 (Copies available from IBM Watson Research Center P O Box 218 Yorktown Heights New York 10598)
33 H G RAYMOND
A queueing theory approach to communication satellite network design
Proc of the International Conference on Communication pp 42-26 to 42-31 1971
34 L G ROBERTS
Multiple computer networks and inter-computer communications
ACM Symposium on Operating Systems Gatlinburg Tenn 1967
35 L G ROBERTS
A forward look
SIGNAL Vol XXV No 12 pp 77-81 August 1971
36 L G ROBERTS
Resource sharing networks
IEEE International Conference March 1969
37 L G ROBERTS
Access control and file directories in computer networks
IEEE International Convention March 1968
38 L G ROBERTS B WESSLER
Computer network development to achieve resource sharing
SJCC 1970 pp 543-549
39 L G ROBERTS B WESSLER
The ARPA computer network
In Computer Communication Networks edited by Abramson and Kuo Prentice Hall 1972
40 R A SCANTLEBURY P T WILKINSON
The design of a switching system to allow remote access to computer services by other computers
Second ACM/IEEE Symposium on Problems in the Optimization of Data Communications Systems Palo Alto California October 1971
41 R A SCANTLEBURY
A model for the local area of a data communication network—Objectives and hardware organization
ACM Symposium on Problems in the Optimization of Data Communication Systems Pine Mountain Ga 1969 pp 179-201

42 R H THOMAS D A HENDERSON
McRoss—A multi-computer programming system
SJCC 1972 In this issue

43 R VAN SLYKE H FRANK
Reliability of computer-communication networks
Proc of the Fifth Conference on Applications of Simulation New York December 1971

44 R VAN SLYKE H FRANK
Network reliability analysis—I
Networks Vol 1 No 3 1972

45 E WOLMAN
A fixed optimum cell size for records of various length
JACM 1965 pp 53-70

46 L KLEINROCK
Scheduling, queueing and delays in time-sharing systems and computer networks
To appear in Computer-Communication Networks edited by Abramson and Kuo Prentice Hall 1972

47 A H WEIS
Distributed network activity at IBM
IBM Research Report RC3392 June 1971

48 M BEERE N SULLIVAN
Tymnet—A serendipitous evolution
Second ACM IEEE Symposium on Problems in the Optimization of Data Communications Systems Palo Alto California 1971 pp 16-20

49 W TEITELMAN R E KAHN
A network simulation and display program
Third Princeton Conference on Information Sciences and Systems 1969 p 29

50 P BURKE
The output of a queueing system
Operations Research 1956 pp 699-704

51 J R Jackson
Networks of waiting lines
Operations Research Vol 5 1957 pp 518-521

52 D B McKAY D P KARP
Network/440—IBM Research Computer Sciences Department Computer Network
IBM Research Report RC3431 July 1971

APPENDIX E

SOME RECENT ADVANCES IN COMPUTER COMMUNICATIONS

by Wesley W. Chu

SOME RECENT ADVANCES IN COMPUTER COMMUNICATIONS*

Author Wesley W. Chu

Organization University of California, Los Angeles, California, U.S.A.

ABSTRACT

Recent advances in computer communications are discussed, including: 1) computer traffic characteristics in the case of long holding time representing scientific applications, and in the case of short holding time representing the inquiry-response systems; 2) telephone channel error characteristics of high speed voice band data transmission on the switched telecommunication network, and of the low speed channel at a rate of 300 bits/sec; 3) optimal fixed message block size for communication systems using error detection and retransmission as error control (with random or burst error channel); 4) statistical multiplexing (Asynchronous Time Division Multiplexing); 5) loop systems and 6) security in computer communications. New areas needing further investigation are included.

I. INTRODUCTION

As computer-communication systems such as time-sharing and distributed computer systems grow in scale and complexity, the problem of being able to understand and to predict system behavior becomes increasingly important. It becomes clear that in order to synthesize a system that meets operating and performance criteria at minimum cost for computing, communication and operation, two key problem areas need to be studied: first, the interfacing problem between computer and communication systems; second, the relationships among communication traffic sources, channels and computer resource allocation mechanisms.

The difficulties in studying and understanding these problems are: 1) computer designers' lack of knowledge in communication technology; 2) communication engineers' lack of knowledge in computer technology; and 3) the lack of tools and models with which to analyse the behavior of these complex systems. The first two difficulties may be resolved by exchange of information between computer designers and communication specialists. The third difficulty may be remedied by periodically summarizing important research related to computer-communication systems which is scattered throughout various journals, conference proceedings and technical reports. In this paper we aim at the last objective.

II. RECENT ADVANCES IN COMPUTER COMMUNICATIONS

1. Computer Traffic Characteristics

It has become apparent that real progress in modeling and analysis depends upon more than elegant analytical results based upon convenient but unsupported assumptions. Measurement and observation are needed; the computer traffic characteristics of in-house time-sharing systems has been undertaken by the Bell Telephone Laboratories to obtain estimates of system variables. Two types of systems under study are: long holding time (connect to disconnect) and short holding time. Long holding time is characteristic of business and scientific applications which require extensive computation; a holding time typically of 15 to 30 minutes. Short holding time is characteristic of inquiry-response systems such as on-line banking, credit bureau and production control which have holding times of a few seconds to one or two minutes.

Jackson and Stubbs [1] and Fuchs and Jackson [2] have reported the results of long holding time. They show that the volume of computer-to-user traffic is an order of magnitude higher than that of user-to-computer traffic. The interarrival time between messages can be approximated by an exponential distribution; that is, the stream of messages can be assumed to constitute a Poisson process. Furthermore, the length of messages can be satisfactorily approximated by the geometrical distribution. During the call interval, the user is active only 5% of the time and the computer is

*This research was supported by the U.S. Office of Naval Research, Mathematical and Information Sciences Division, Contract No. N00014-69-A-0200-4027, NR 048-129 and the Advanced Research Projects Agency of the Department of Defense under Contract No. DAHC 15-60-C-0285.

active about 30% of the time. Thus, the channel is idle for a significant portion of the holding time. The traffic characteristics of short holding time are reported on by Dudick, Fuchs and Jackson [3]. The measured results from four such systems reveal that user send time (the total amount of time during which user characters are being transmitted) is less than 15% of the holding time. This parameter is important to the design of statistical multiplexors. The character interarrival times can be represented as a sum of two gamma distributions; the number of user segments per call and computer segments per call, can be represented by a geometrical distribution. These measurements and estimated system variables not only provide us with insight into the behavior of the system and shed light on areas that need improvements, but are essential in the modeling and analysis of computer communication systems.

2. Channel Error Characteristics

The communication channel provides the links between processors and terminals and plays an important role in computer communication systems. Thus, a characterization of channel performance is important for understanding the cause of errors, for suggesting the possible improvements in the design of transmission equipment and the design of efficient error control systems, and for planning optimal computer communication systems.

To characterize telephone channel error performance, a survey measurement program of the telephone network to determine the error performance and the data speed capabilities is necessary. A series of such studies started by the Bell Systems in 1958 was directed towards this goal, and several papers have been published on this subject [4-8]. Here we shall emphasize recent survey results (1969-1970) on high speed voice band data transmission performance on the switched telecommunication network [7] and low speed data transmission performance on the switched telecommunication network [8]. In these two papers, the distributions of error per call are given on a bit, burst and block basis. Information is also presented on the distribution of intervals between errors, the structure of burst errors and the number of errors in blocks of various sizes. Such statistics provide information on channel reliability and are also useful in the design of efficient error control procedures.

In the high speed voiceband data transmission channel, toll traffic was used as a basis for the sampling plan which resulted in the selection of approximately 600 dialed-toll connections between geographically dispersed local switching offices. Data rates of 1200, 2000, 3600 and 4800 bits/sec are measured on the Bell System switch telecommunication network. The measured results show a substantial improvement of performance in comparison with the results of previous surveys; for example, the measured results for operation at 1200 and 2000 bits/sec show that approximately 82% of the calls have an average error rate of 1 error in 10^5 bits or better over short, medium and long haul calls; while the 1959 Alexander, Gryb and Nast survey [4] shows only 63% of their test calls reached this performance level (10^{-5}) for operation at the same data rate. A general tendency for performance to degrade with transmission distance has been noted. These results also indicate that impulse noise accounted for a large percentage of the observed errors.

Low speed data transmission corresponds to teletypewriters, computer ports and other terminal devices that communicate by means of data organized in characters (comprised of several bits) using start-stop transmission at a rate of 300 bits/sec. Character error statistics, rather than bit error statistics, are the parameters of interest in this type of transmission because the message consists of a display (in teletypewriters) or use characters in most applications (in computers). Measurements were made on 534 connections with over 21 million characters (1 character = 10 bits) transmitted. Over 90% of the low speed test calls contained about 36,000 to 54,000 characters. A character error rate of 10^{-4} or less is indicated for approximately 78% of all calls, while 95% of all calls have a lost character rate of 10^{-4} or less. Errors occurring in the messages are in bursts rather than at random. The number of character errors in a block increases with the block length. Since this is the first report on low speed data, no comparison with any previous survey is possible. Further analysis of the statistics will give insight into the causes of errors which in turn may suggest approaches to improve error performance.

3. Optimal Fixed Message Block Size

The message outputs from a computer are usually in strings of characters or bursts. The variation of message length can best be described by a probability distribution. For ease in data handling and memory management, the random message length is usually partitioned into several fixed size blocks. Due to the random length of the message, the last partitioned block usually is not filled by the message but is filled with dummy information.

For reasons of economics and reliability, error detection and retransmission is employed in many data communication systems [5, 9]. The optimal block size is an important parameter in the design of such systems. From the acknowledgement point of view, it is desirable to select the largest possible block size. Since each message block requires at least one acknowledgement signal, the fewer the number of blocks needed for a message, the [illegible] the channel capacity required for acknowledgements. On the other hand, since a larger message block has a higher channel wastage due to the last unfilled partitioned block, and also has a higher probability of error, it is desirable to select the smallest possible block size. Thus there is a trade-off in selecting the optimal block size.

Kucera [10], Balkovic and Muench [11] and Kirlin [12] have studied the optimal message block size for the error detection and retransmission system that maximises transmission efficiency. Chu [13] considers an additional important parameter - the message (file) length - in determining the fixed message block size, which significantly effects the selection of the optimal block size. His model considers average message (file) length, message length distribution, channel error characteristics (random error and burst error), overhead for addressing, error control and acknowledgement delay. His criterion for optimality is to minimize the time wasted in acknowledgements, retransmissions and the waste in the last unfilled block.

4. Statistical Multiplexing

Multiplexing is commonly used to share and to efficiently utilize a communication channel. Currently data multiplexing has taken two forms: Frequency Division Multiplexing (FDM) and Synchronous Time Division Multiplexing (STDM) commonly known as time division multiplexing. Frequency Division Multiplexing divides the channel bandwidth into several subchannels such that the bandwidth of each subchannel is greater than that required for a message channel. Because of the need to employ "guard bands" to prevent data signals from each of the data channels from feeding into adjacent channels and because of the relatively poor data transmission characteristics of the voice band channel near the edges of its bandwidth, FDM does not make as efficient use of the voice band as does STDM.

In STDM, each user (terminal) is assigned a fixed time duration or time slot on the communication channel for the transmission of messages from terminals to computer. The multiplexing apparatus scans the set of users in a round robin fashion. After one user's time duration has elapsed, the channel is switched to another user. With appropriately designed synchronous operation, required buffering can be limited to one character per terminal. Addressing is usually not required since the user is identified by the time slot position. The STDM technique, however, also has certain disadvantages. It is inefficient in channel utilisation to permanently assign a segment of bandwidth that is utilized only a portion of the time. Statistics collected from several typical operating time sharing systems [1] showed that during a call (connect to disconnect), the user-to-computer traffic for the long holding time case is active only 5% of the time. Thus STDM would be very inefficient in channel utilization in such an environment since STDM allocates a time slot to each user independent of his activity. In order to increase channel utilisation, statistical multiplexing or Asynchronous Time Division Multiplexing (ATDM) has been proposed [14, 15] for computer communications. The basic idea is to switch from one user to another user whenever the former is idle and the latter ready to transmit data. Thus the data is asynchronously or statistically multiplexed with respect to the users. With such an arrangement, each user would be granted access to the channel only when he has a message to transmit. The crucial attributes of such a multiplexing technique are: 1) an address is required for each transmitted message and 2) buffering is required to handle statistical fluctuations in the input traffic.

The data structure for messages forming the input to the multiplexor buffer can be classified into four categories: constant length messages; random length messages; mixed (constant and random length) messages; random length message segmented into fixed size blocks. The constant length message input corresponds to teletype (TTY) input, each user types in one character at a time. The random length message input corresponds to paper tape input, cathode-ray tube (CRT) input or computer output. The mixed message input corresponds to traffic from a mixture of types of input terminals such as CRT, TTY, etc. For ease in data handling and memory management, random length messages are often segmented into fixed size blocks which correspond to the last type of data structure. Since messages have random length, the last block of a message usually cannot be entirely filled. As a result, for a given traffic intensity, this type of data structure requires a larger buffer than that of random length messages which are not segmented [14]. The buffer behavior of these four types of data structures have been analysed by finite waiting room queueing models [14-17].

The buffer behavior of a statistical multiplexor for mixed message inputs lies between that of constant length and random length messages [17]. The output process of a statistical multiplexor has been studied by Pack [18].

The demultiplexor distributes messages to appropriate destinations according to their message addresses. Thus, the behavior of the demultiplexing buffer not only depends on traffic intensity but also on traffic scheduling to various destinations. In the case of a time sharing system, message scheduling is determined by the job scheduling algorithm of the computer operating system. In the case of distributed computer systems, message scheduling is influenced by the message routing algorithm. A simulation study of the demultiplexing buffer behavior revealed that, for a given input traffic volume, the best buffer behavior can be achieved by scheduling an equal amount of traffic to each destination [19]. Hence there is a close relationship among demultiplexing system performance and scheduling algorithms [20] in computer operating systems and/or message routing algorithms. Further research in this direction would be desirable. Results obtained in this area will be essential in the joint optimisation of the overall performance of such computer communication systems.

Buffering is required to provide error control and message scheduling which are two important functions in computer communication systems. Since statistical multiplexing requires buffering to

handle statistical fluctuations, the multiplexing buffer can also be used for these functions.

From these studies we conclude that in an ATDM system, an acceptable buffer overflow probability can be achieved by a reasonable buffer size; the expected queueing delay is very small and acceptable for most applications. Hence, ATDM or statistical multiplexing is a feasible technique for data communications. Furthermore ATDM greatly improves the transmission efficiency and system organization. We have constructed a statistical multiplexor at UCLA. Our preliminary experience has shown that the gain in communication cost, especially in long distance transmission, by employing ATDM in computer communication could far outweigh costs of overhead in addressing and storage for buffering. Statistical multiplexing should, therefore, have high potential for use in future computer-communication systems.

5. Loop Systems

A special distributed computer system architecture of considerable recent interest is the loop (ring) system. This type of system connects all terminals and/or computers by a common bus or loop. The major advantages are the simple routing algorithm and ease in control of information. Farmer and Newhall [21] proposed and constructed a loop system with bursty traffic which connects various devices such as teletype, plotter, cathode-ray tube display, disk control unit and computer together. Yuen, et al. [22] presents some approximate results on the traffic behavior of this distributed loop system. Pierce [23] proposed a hierarchy of interconnected loop systems with random length messages segmented into fixed size blocks, and provides a scheme for transferring information among the various levels of loops. Konheim and Meister [24] analyzed such a hierarchical system. Hayes and Sherman [25] studied the message delay due to buffering for a single loop system. The data source is assumed to be of a bursty nature. The traffic generated by each user is assumed to be identically distributed with uniformly distributed destinations. Konheim and Meister [26] studied the loop system as a priority service system. Messages may enter the system at any input terminal located on the loop. Priority is assigned on the basis of position on the loop; the terminal closest to the computer has highest priority and priorities decrease with 'distance' from the computer. The stationary queue lengths and average virtual waiting time are calculated. Spragins [27] has calculated the waiting time of this priority loop system with Poisson arrival process. Kaye [28] has obtained the mean and variance of message waiting time and proportions of blocked messages of fixed length messages of a loop system. Farber and Larson [29] have proposed a loop system using fixed message length similar to that of Pierce except messages are addressed to processes rather than processors. Further, messages can be only removed at their origin.

Research results have shown that the acceptable queueing delay can be achieved when the traffic in a loop system is under careful control. One of the shortcomings of such a system however, is system reliability. Further study is needed in determining the reliability performance of loop systems and ways of improving system reliability. Such studies will enable us to compare the cost-performance of loop systems with that of other systems.

6. Security in Computer Communications

With the growth in the use of remote terminal devices and time-sharing systems in making information available to a wide variety of users for widely diverse applications, the problem of computer communication security becomes increasingly important [30-32]. The communication channels are perhaps the most vulnerable components of the computer system because they are easy to access by methods such as wire tapping, picking up electromagnetic pulses, or "piggy-back" entry [32]. The nature of computer communication security is quite different from that of the classical communication security. They differ at least in the following ways. 1) Computer files usually offer a large amount of data to work on. The enemy would have to know exactly what type of information was in each file (e.g., programs, address files, scientific information) in order to steal it. 2) In computer files, all records are usually similar. Programs have a high rate of repeated characteristics (e.g., COBOL, FORTRAN). Also, quite often the structure of the computer program can be guessed at. All these similarities can help an enemy cryptanalyst to decode even when fairly sophisticated cryptographic techniques are used. Thus new methods are needed to provide security in computer communications.

The use of logic operation (e.g., exclusive OR) and pseudo-random numbers presently seems to offer the greatest possibility for computational cryptography. A unique key for each message is generated from the pseudo-random number generator, and the message then performs logical (exclusive OR) operations with the key to produce the cipher message. The reverse operation deciphers the cipher into its original message. This type of operation is usually very fast and efficient in modern computers and has the advantage of good security, low cost, and easy changeability. These methods have been explored by Skatrud [33], Krishnamurthy, [34] and Tassel. [35]

To further increase security in computer information, we could introduce multiple pseudo-random number generators to achieve higher levels of security. For example, the system could use one pseudo-random number generator to cipher the message; then each user could further provide their own pseudo-random number generator to provide a second level of ciphering. Another way, perhaps even more effective, is to use an expanded character set technique to break down the statistical parameters of the message, such as frequency of

single letters, diagrams, vowel percentages, etc. before performing the logical operation with the pseudo-random number. Our preliminary results indicate this scheme provides excellent security performance. Detailed findings will be reported in the near future.

Sharing of files and data bases is not only important in applications, but also greatly increases flexibility and computer capability. Security and protection are absolutely necessary in these facilities. The full utilization of these systems relies heavily on efficient techniques to provide effective information security. Much work still needs to be done.

Other advances which have not been included in this paper but have impact on computer communications are: error control, modulation, transmission medium, social and regulatory policies, and standardization.

REFERENCES

1. P. E. Jackson and C. D. Stubbs, "A Study of Multi-access Computer Communications," AFIPS Conference Proceedings, Vol. 34, 1969, pp. 491-504.

2. E. Fuchs and P. E. Jackson, "Estimates of Distributions of Random Variables for Certain Computer Communications Traffic Models," CACM, Vol. 13, No. 12, Dec. 1970, pp. 752-757.

3. A. L. Dudick, E. Fuchs and P. E. Jackson, "Data Traffic Measurements for Inquiry-Response Computer Communication Systems," Proc. IFIP Congress 1971, Ljubljana, Yugoslavia, August 1971.

4. A. A. Alexander, R. M. Gryb and D. W. Nast, "Capabilities of the Telephone Network for Data Transmission," BSTJ, Vol. 39, No. 3, May 1960, pp. 431-476.

5. R. L. Townsend and R. N. Watts, "Effectiveness of Error Control in Data Communications Over the Switched Telephone Network," BSTJ, Vol. 43, No. 6, November 1964, pp. 2611-2638.

6. C. W. Farrow and L. N. Holsman, "Nationwide Field Trial Performance of a Multilevel Vestigial Sideband Data Terminal for Switched Network Voice Channels," Conference Records, 1968 IEEE Conference Communication, Philadelphia, Pennsylvania, June 1968, p. 782.

7. M. D. Balkovic et al., "1969-70 Connection Survey: High-Speed Voiceband Data Transmission Performance on the Switched Telecommunications Networks," BSTJ, Vol. 50, No. 4, April 1971, pp. 1385-1405.

8. H. C. Fleming and R. N. Hutchinson Jr., "1969-70 Connection Survey: Low-Speed Data Transmission Performance on the Switched Telecommunications Networks," BSTJ, Vol. 50, No. 4, April 1971, pp. 1385-1405.

9. S. Y. Tong, "A Survey of Error Control Techniques on Telephone Channels," Proceedings of the 1970 National Electronic Conference, Chicago, pp. 462-476.

10. J. J. Kucera, "Transfer Rate of Information Bits," Computer Design, June 1968, pp. 56-59.

11. M. D. Balkovic and P. E. Muench, "Effects of Propogation Delay Caused By Satellite Circuits on Data Communications Systems that Use Block Retransmission for Error Correction," Conference Record IEEE Conference on Communications, Boulder, Colorado, June 1969, pp. 29/31-29/36.

12. R. L. Kirlin, "Variable Block Length and Transmission Efficiency," IEEE Trans. Comm. Tech., Vol. 17, No. 3, June 1969, pp. 350-354.

13. W. W. Chu, "Optimal Fixed Message Block Size for Computer Communications," Proc. IFIP Congress 1971, Ljubljana, Yugoslavia, August 1971.

14. W. W. Chu, "Design Considerations of Statistical Multiplexors," Proc. ACM Symposium on Problems in the Optimization of Data Communications Systems, Pine Mountain, Georgia, 1969, pp. 39-60.

15. W. W. Chu, "A Study of Asynchronous Time Division Multiplexing for Time-Sharing Computers," AFIPS Conference Proceedings, Vol. 35, 1969, pp. 669-678.

16. W. W. Chu, "Buffer Behavior for Batch Poisson Arrivals and Single Constant Output," IEEE Trans. Comm. Tech., Vol. 18, No. 5, October 1970, pp. 613-618.

17. W. W. Chu and L. C. Liang, "Buffer Behavior for Mixed Input Traffic and Single Constant Output Rate," IEEE Trans. Comm. Tech., April 1972, pp. 230-235.

18. C. D. Pack, "The Effect of Multiplexing on a Computer Communication System," submitted to the Comm. ACM.

19. W. W. Chu, "Demultiplexing Considerations for Statistical Multiplexors," IEEE Trans. Comm. Tech., June 1972.

20. L. Kleinrock, "Scheduling, Queueing and Delays in Time-Shared Systems and Computer Networks," Chapter 4 of Computer-Communication Networks, edited by N. Abramson and F. Kuo, Prentice Hall, 1973.

21. W. D. Farmer and E. E. Newhall, "An Experimental Distributed Switching System to Handle Bursty Computer Traffic," Proc. ACM Symposium on Problems in the Optimization of Data Communications Systems, Pine Mountain, Georgia, October 1969, pp. 1-35.

22. M. L. T. Yuen, et al., "Traffic Flow in a Distributed Loop Switching System," PIB International Symposium XXII on Computer Communication Networks and Teletraffic, April 1972.

23. J. P. Pierce, C. H. Cohen and W. J. Kropfl, "Network for Block Switching of Data," IEEE Conference Records, New York, March 1971.

24. A. G. Konheim and B. Meister, "Waiting Lines in Multiple Loop Systems," to appear J. Math. Analysis and Applications.

25. F. J. Hayes and D. N. Sherman, "Traffic Analysis of a Ring Switched Data Transmission System," BSTJ, Vol. 50, No. 9, November 1971, pp. 2947-2978.

26. A. G. Konheim and B. Meister, "Service in a Loop System," to appear in JACM 1972.

27. J. D. Spragins, "Loops Use for Data Collection," PIB International Symposium XXII on Computer-Communication Networks and Teletraffic, April 1972.

28. R. Kaye, "Analysis of a Distributed Control Loop for Data Communication," PIB International Symposium XXII on Computer Communication Networks and Teletraffic, April 1972.

29. D. Farber and K. C. Larson, "The System Architecture of the Distributed Computer System - The Communication System," PIB International Symposium XXII on Computer Communication Networks and Teletraffic, April 1972.

30. P. Baran, "Communications, Computers, and People," AFIPS Conference Proceedings, 1965 Spring Joint Computer Conference, Vol. 27, pp. 45-49.

31. W. H. Ware, "Security and Privacy in Computer Systems," AFIPS Conference Proceedings, 1967 Spring Joint Computer Conference.

32. H. E. Peterson and R. Turn, "System Implications of Information Privacy," AFIPS Conference Proceedings, 1967 Spring Joint Computer Conference, Vol. 30, pp. 291-300.

33. R. D. Skatrud, "A Consideration of the Application of Cryptographic Techniques to Data Processing," AFIPS Conference Proceedings, 1969 Fall Joint Computer Conference, Vol. 35, pp. 111-117.

34. E. V. Krishnamurthy, "Computer Cryptographic Techniques for Processing and Storage of Confidential Information," International Journal of Control, November 1970.

35. D. V. Tassel, "Advanced Cryptographic Techniques for Computers," Communications of the ACM, December 1969, Vol. 12, No. 12, pp. 664-665.

APPENDIX F

RAND SATURATION EXPERIMENT

PRELIMINARY RESULTS

by V. Cerf and W. Naylor

Network Measurement Note #2
NIC #10352

V. Cerf
W. Naylor
7 May 72

RAND SATURATION EXPERIMENT

PRELIMINARY RESULTS

VINTON G. CERF
WILLIAM NAYLOR

7 MAY 1972

NETWORK MEASUREMENT CENTER
UNIVERSITY OF CALIFORNIA, LOS ANGELES
3804 Boelter Hall
Computer Science Department
Los Angeles, Calif. 90024

RAND SATURATION EXPERIMENT: PRELIMINARY RESULTS

SUMMARY

This experiment is designed to study delay and throughput characteristics of a part of the ARPA network when a single node is under increasing background loading. In particular, RAND's IMP is loaded by activating IMP message generator fake HOSTs at selected IMPs in the net whose messages are destined for the RAND IMP fake HOST discard.

The delay and throughput experienced by a single site, UCLA, under varying loading conditions, are measured and the results discussed in the light of known characteristics of the IMP-IMP message switching protocols (see also references BBN1822 and HEART1970).

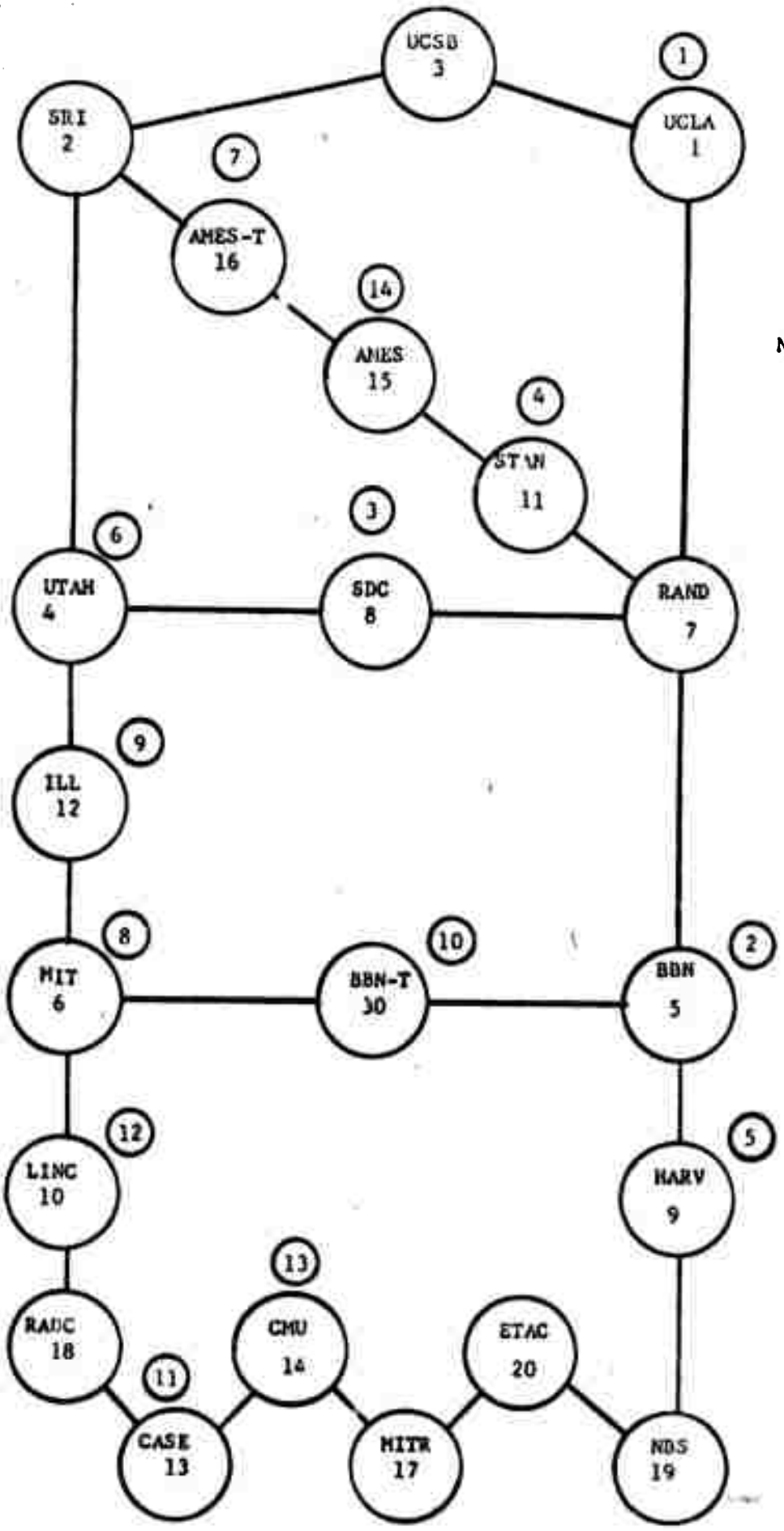
EXPERIMENT DESIGN

At the time this experiment was performed, RAND was the only site in the net at which four 50KB telephone lines were connected to a single IMP (see map on next page). It was possible, then, to transmit as much as 200KB into RAND's IMP over these four lines.

The artificial traffic generators in each IMP are capable of transmitting messages to a single site over a single logical link. The messages may be transmitted with deterministic inter-departure times, and the minimum allowable delay between messages is the time for the message to be sent and a RFNM (Request for Next Message) to be returned to the sending HOST. This minimum delay is on the order of 30 mseconds for adjacent sites and increases as the number of IMPs between source and destination IMPs increases. Message lengths can range from 0 to about 8000 bits (exclusive of leader and padding). Details of the IMP-IMP protocols and message generators can be found in (BBN1822).

There are a number of parameters which can be varied in an experiment of this kind; among these are message lengths, number of message generators running, message inter-departure times, topology of source and destination IMP data paths, etc. In this experiment, the nodes of the network are divided into four classes: measured (UCLA IMP), receiving node (RAND IMP), senders, unused. Messages are sent from UCLA and from other sending nodes to RAND. Delays and throughput are measured for UCLA only.

88



Note: Small circled numbers show order in which traffic generators were turned on to increase traffic flow to RAND. Numbers inside site circles represent ARPANET site numbers.

ARPA NETWORK TOPOLOGY DURING SATURATION EXPERIMENT

The length of messages sent from UCLA or other sending sites to RAND varied in length from 1 packet/ message (960 bits) to 4 or 8 packets/message (3840 and 7936 bits, respectively). Table 1 shows the various combinations of message lengths used in the 9 different experiments.

UCLA	OTHER SENDERS	
1 p/m	1 p/m	p/m = packets/message
1	4	
1	8	
4	1	
4	4	
4	8	
8	1	
8	4	
8 p/m	8 p/m	

Message length combinations in packets/message

Table 1

For each combination of message lengths, traffic is first turned on and measured from UCLA to RAND (see ARPANET map on page 2). This is accomplished by turning on both the artificial traffic generator at UCLA's IMP and a cumulative statistics package also in UCLA's IMP which accumulates statistics on delays and throughputs for traffic leaving UCLA's IMP. All message generators are run with RFNM driven traffic (i.e. minimum inter-departure times). Statistics are gathered for six minutes (30 samples of 12 second cumulative statistics), and then a new IMP message generator is started for the next six minute period (previous generators are left running). In the map on page 2, the order in which IMP message generators are activated for increasing number of senders is shown by a small circled number next to a site.

EXPERIMENT RESULTS

In the figures which follow, the results of the experiment are shown; one important point to keep in mind is that the delay and throughput characteristics observed are specifically for a source IMP (UCLA) which is only one "hop" away from the destination IMP (RAND) and that these results are expected to differ for a measured site which is not adjacent to the destination (i.e. more than one "hop" away). Future experiments will investigate the variation in throughput and delay for increasing hop distance between source and destination nodes.

To aid the reader, we define a few terms which are used in the figures below. TOTAL THROUGHPUT is the number of bits/second actually sent out from the measured site (UCLA). This traffic includes retransmissions and overhead control bits, some of which may not have been accepted by the destination site because there was not enough buffer space to hold the data when the packet arrived. EFFECTIVE THROUGHPUT is computed by multiplying the number of RFNM's received by the artificial traffic generator at UCLA by the length of the messages being sent. Effective throughput does not include overhead bits and retransmissions. ROUND-TRIP DELAY is the delay between the departure of a message from the artifical traffic generator to the time a RFNM is received for that message. For multi-packet messages, RFNM's are sent only after all packets have arrived at the destination site and have started into the receiving HOST (the fake HOST discard at RAND). The DELAY/PACKET is computed by dividing the Round-trip delay by the message length in packets.

The first three figures below consider the results for the case that UCLA sends only single packet messages to RAND while other sites send 1, 4, or 8 packet messages. When all sites are sending single packet messages, the total throughput shown in figure 1 tends to drop off slightly with increasing interference. As shown in figure 2, the effective throughput follows this curve, but is slightly below due to the small overhead of each message. The corresponding delay, shown in figure 3, increases slightly.

For the case that other sites send 4 or 8 packet messages while UCLA sends single packet messages, there is much more noticeable interference. In the case of 8 packet message interference, comparison of the total and effective throughput graphs (figures 1 and 2) shows that after 5 sites are sending, UCLA begins to experience retransmissions as well as simple packet overhead. This shows up in figure 3 as a non-linear increase in delay/packet.

The maximum effective data rate sustained by a single link is shown in figure 2 as roughly 27.5 KBS (kilobits per second).

Figures 4, 5, 6 consider the case that UCLA is sending 4 packet messages. We notice in figure 5 that the maximum single link data rate is now about 37 KBS. The increase is not more since the RFNM delay has not decreased to one fourth of its single packet message value, but is only three fourths (26./34.). However, there is a drastic interference between multi-packet messages, as we see in figure 4. After more than 3 sites are sending multi-packet messages, the total throughput drops rapidly. The effective throughput drops even faster (see figure 5). The peculiar shape of the curve in figure 4 for UCLA sending 4 packet messages and others sending 8 (i.e. total throughput curve increases again), can be explained by the following argument. Total throughput begins to drop as interference increases because there is contention for buffer space and the RFNM delay increases. After a while, however, the sending site begins to time-out while waiting for an ACK from transmitted packets, and starts to re-transmit. Retransmissions, of course, are not subject to the "blocked link" convention, and thus can create traffic in excess of the maximum achievable effective traffic. (Figure 1 also shows this effect).

The sudden drop in throughput (total and effective) after more than three multi-packet senders are running is the result of contention for re-assembly buffer space in RAND's IMP. There is enough re-assembly buffer space for three full multi-packet messages. The buffers are allocated on the assumption that any incoming multi-packet belongs to a message containing 8 packets, since it is not known at allocation time how long the multi-packet really is. When more than 3 sites send multi-packet messages to RAND's IMP, there is strong contention for the available buffer space.

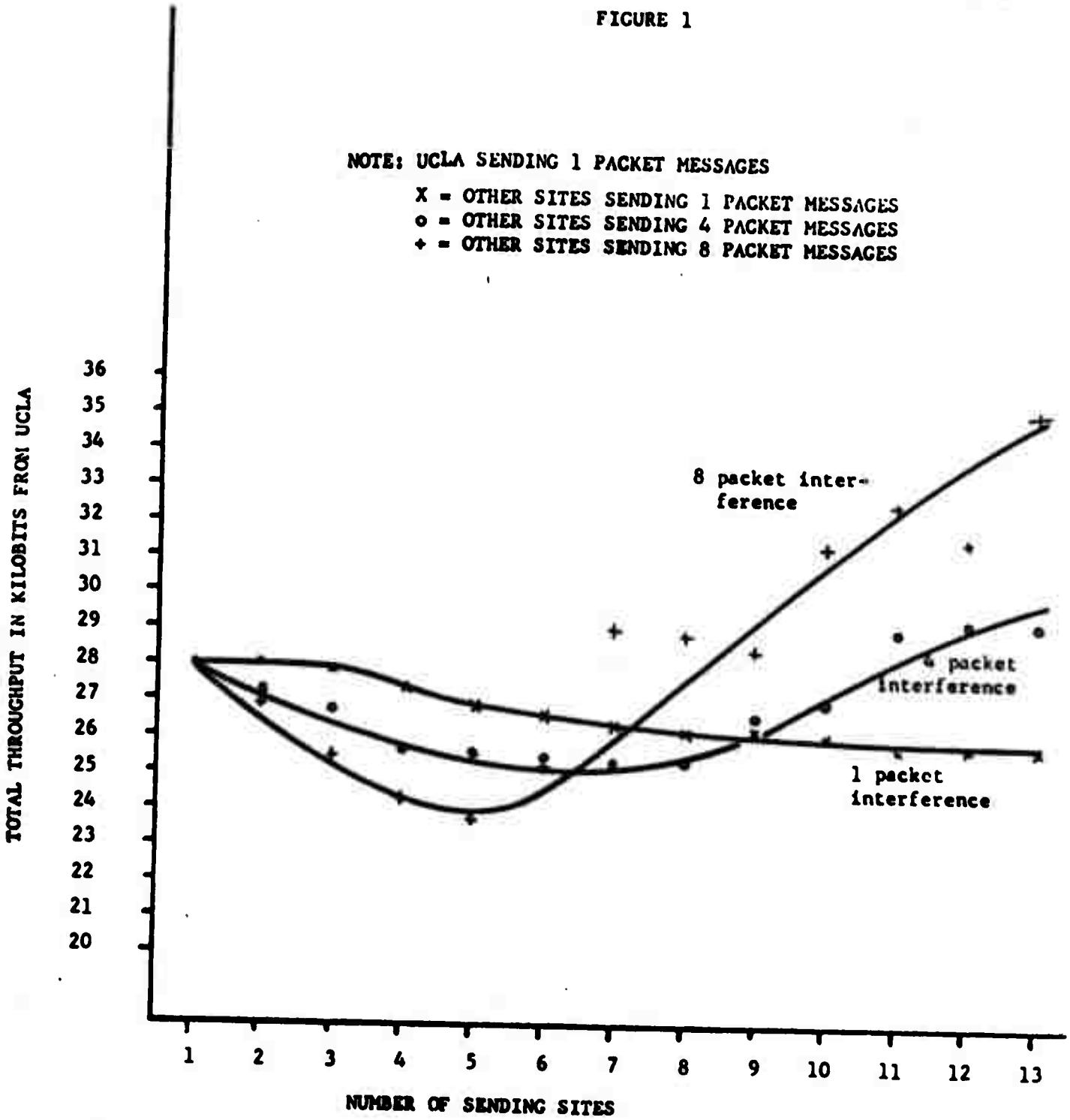
Similar results are shown in figures 8, 9, and 10 for the case that UCLA sends full 8 packet multi-packet messages. The maximum sustainable effective throughput fo: a single link is 40.5 KBS. We note that when there is little or no interfereing traffic, the RFNM delay/packet drops from 34 msec to 25 msec as message length increases from 1 to 8 packets.

TOTAL THROUGHPUT FROM UCLA VERSUS NUMBER OF SENDING SITES

FIGURE 1

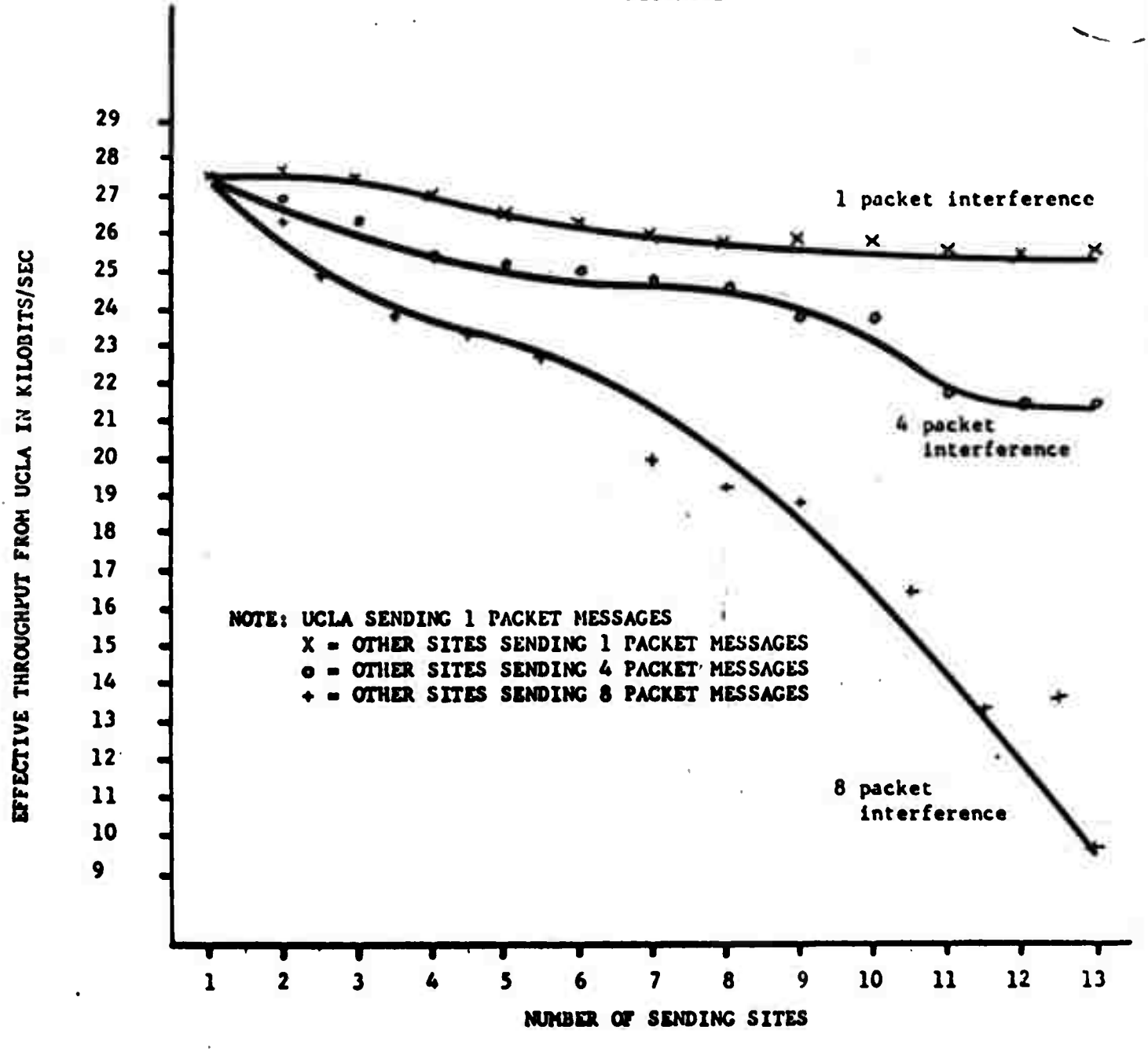
NOTE: UCLA SENDING 1 PACKET MESSAGES

X = OTHER SITES SENDING 1 PACKET MESSAGES
o = OTHER SITES SENDING 4 PACKET MESSAGES
+ = OTHER SITES SENDING 8 PACKET MESSAGES



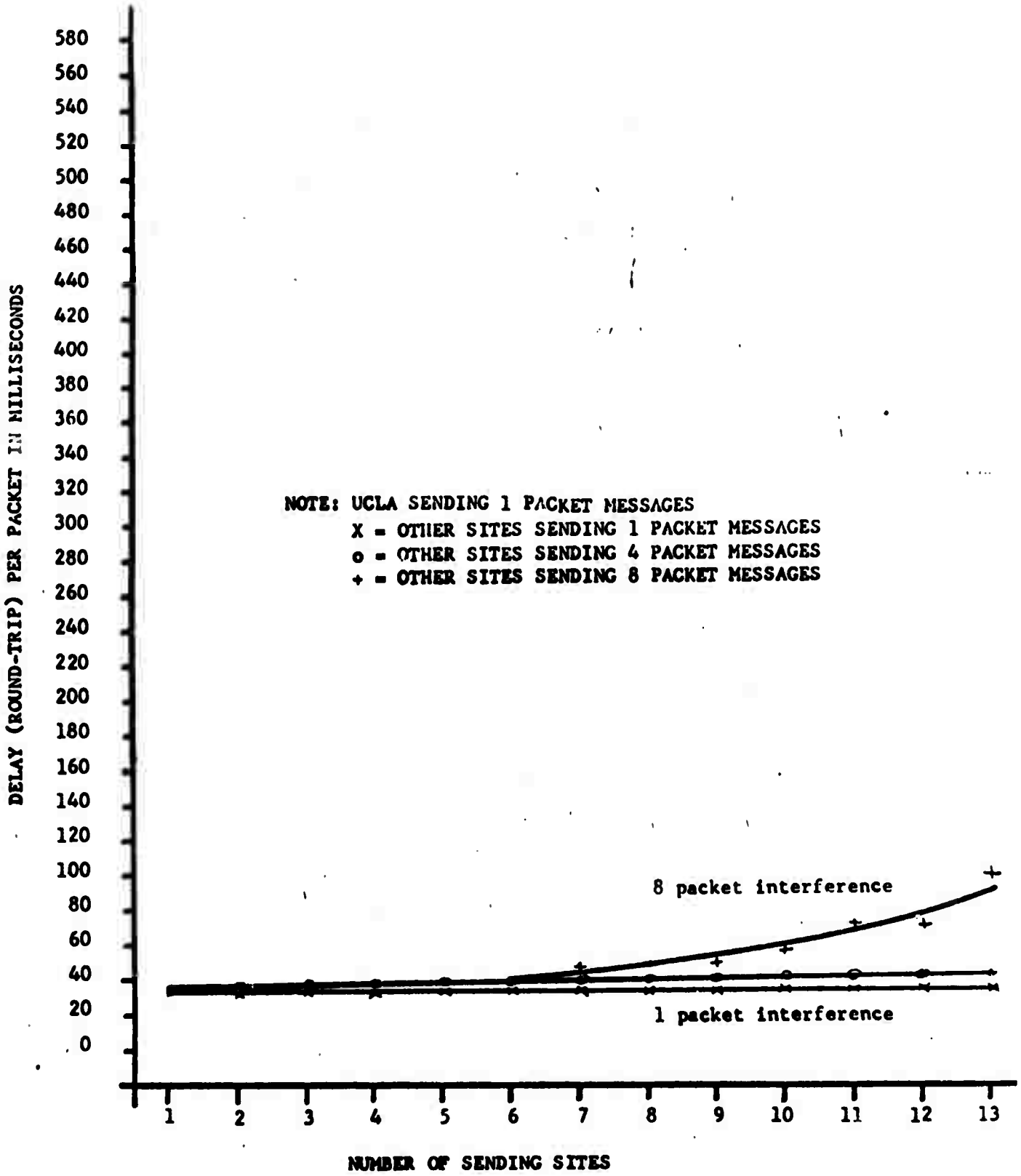
EFFECTIVE THROUGHPUT FROM UCLA VERSUS NUMBER OF SENDING SITES

FIGURE 2



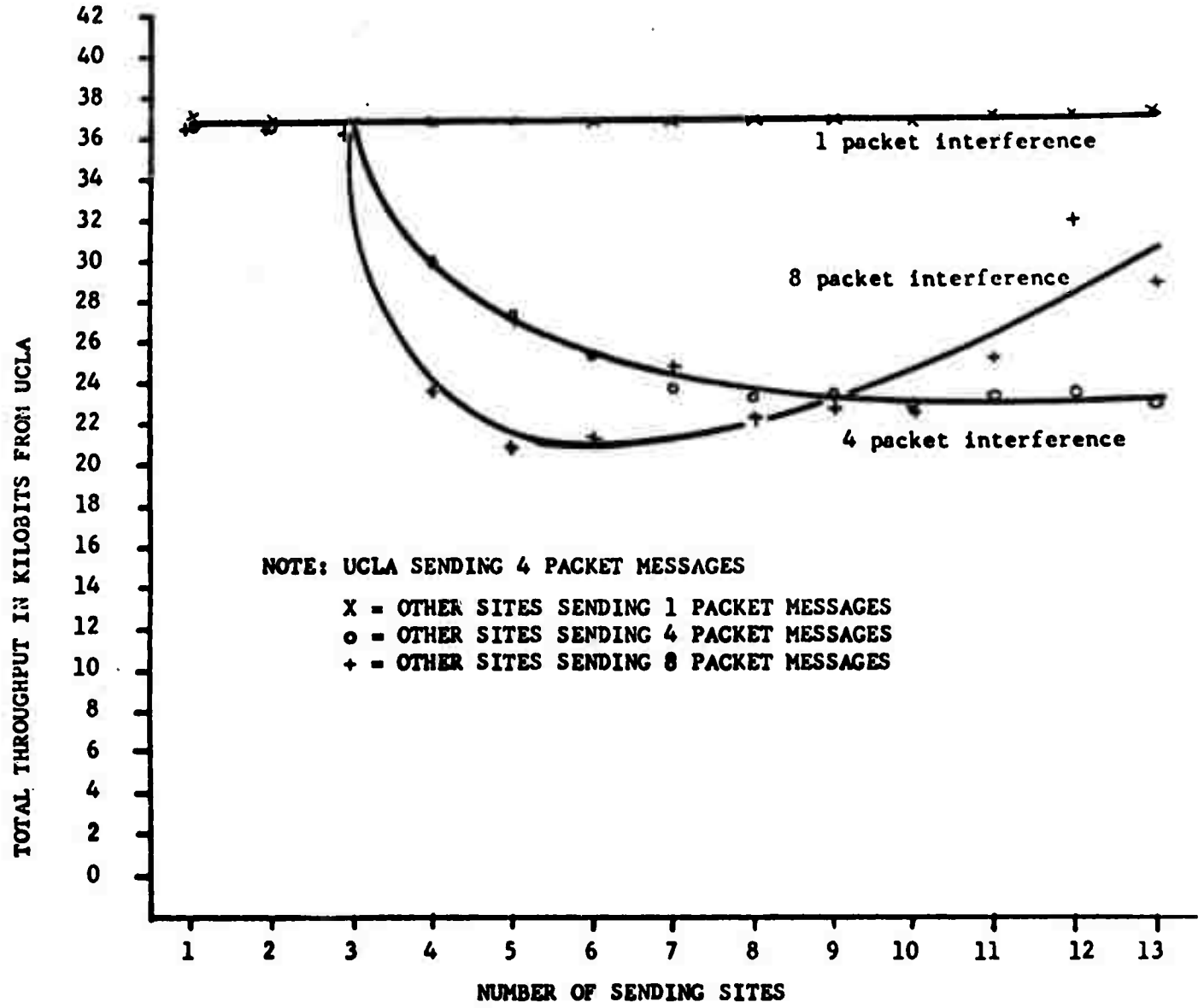
DELAY AT UCLA VERSUS NUMBER OF SENDING SITES

FIGURE 3



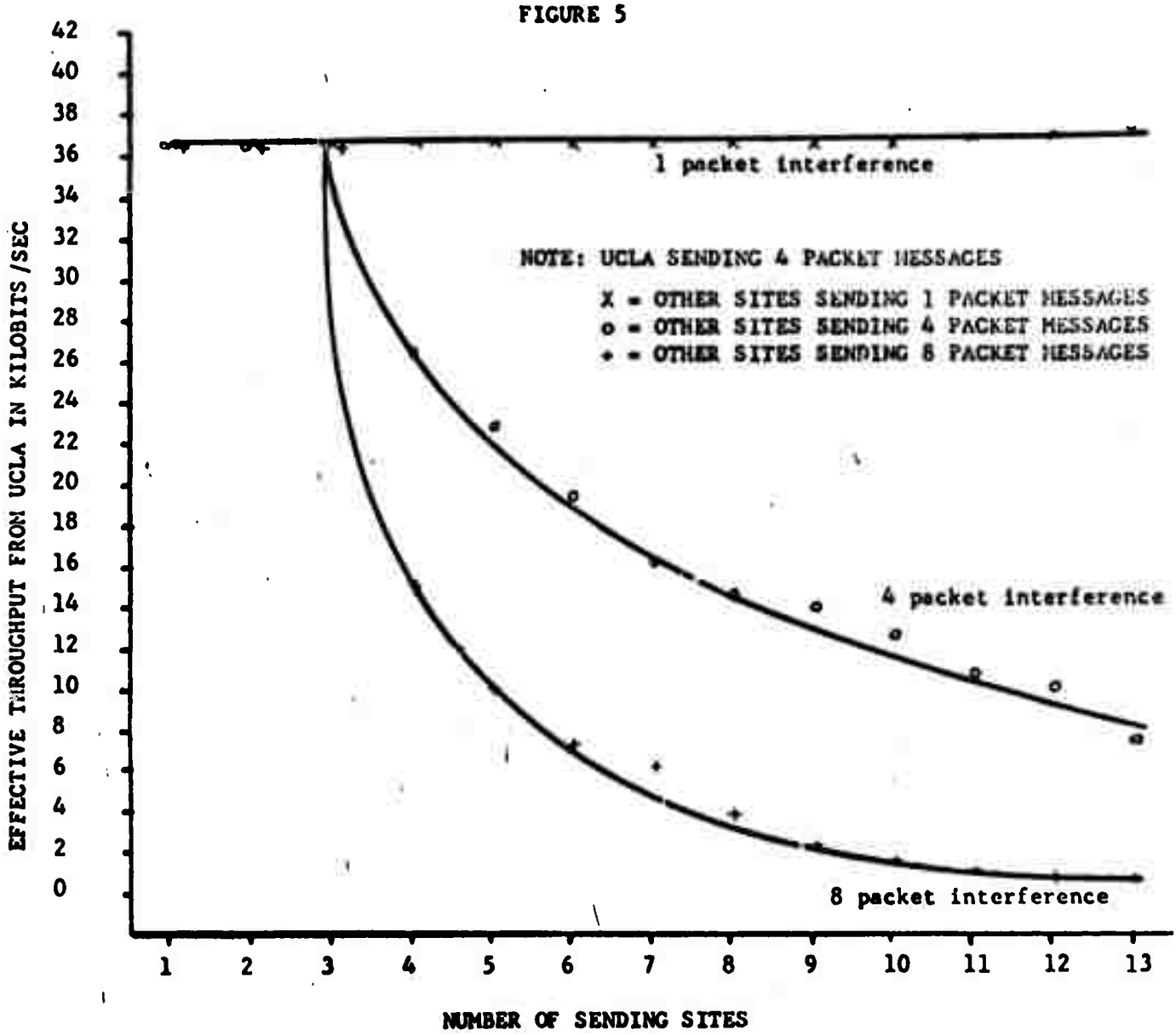
TOTAL THROUGHPUT FROM UCLA VERSUS NUMBER OF SENDING SITES

FIGURE 4



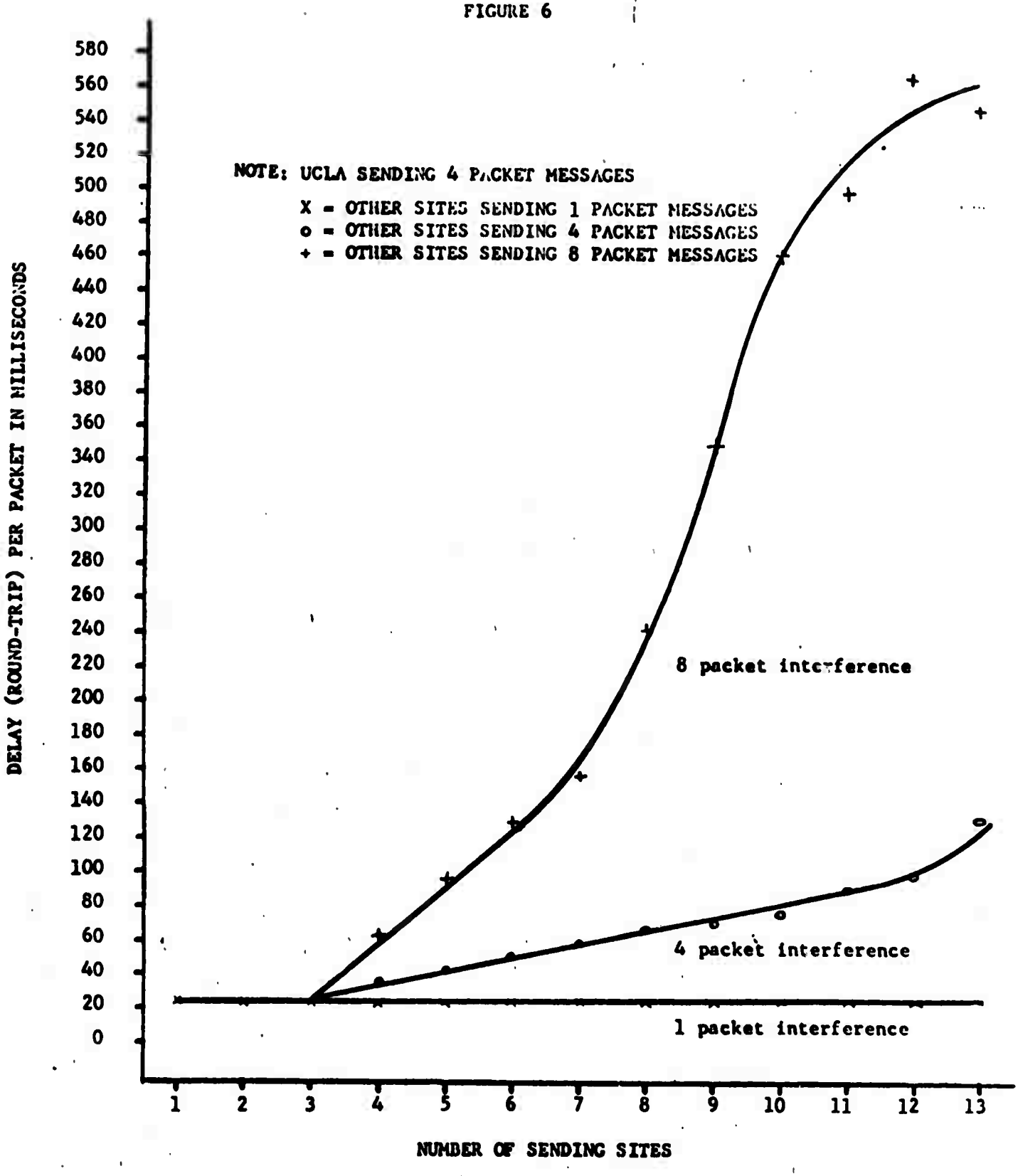
EFFECTIVE THROUGHPUT FROM UCLA VERSUS NUMBER OF SENDING SITES

FIGURE 5



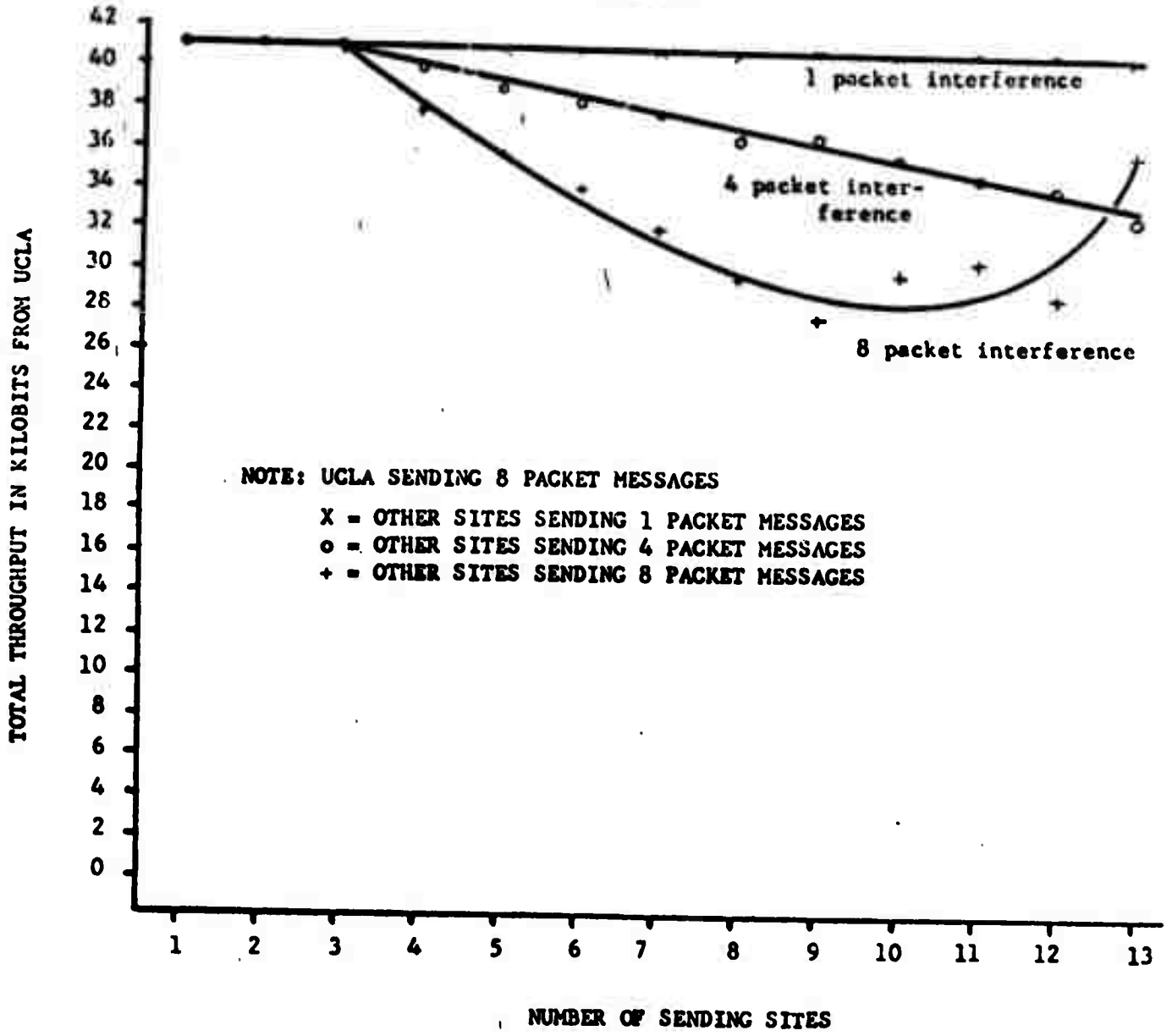
DELAY AT UCLA VERSUS NUMBER OF SENDING SITES

FIGURE 6

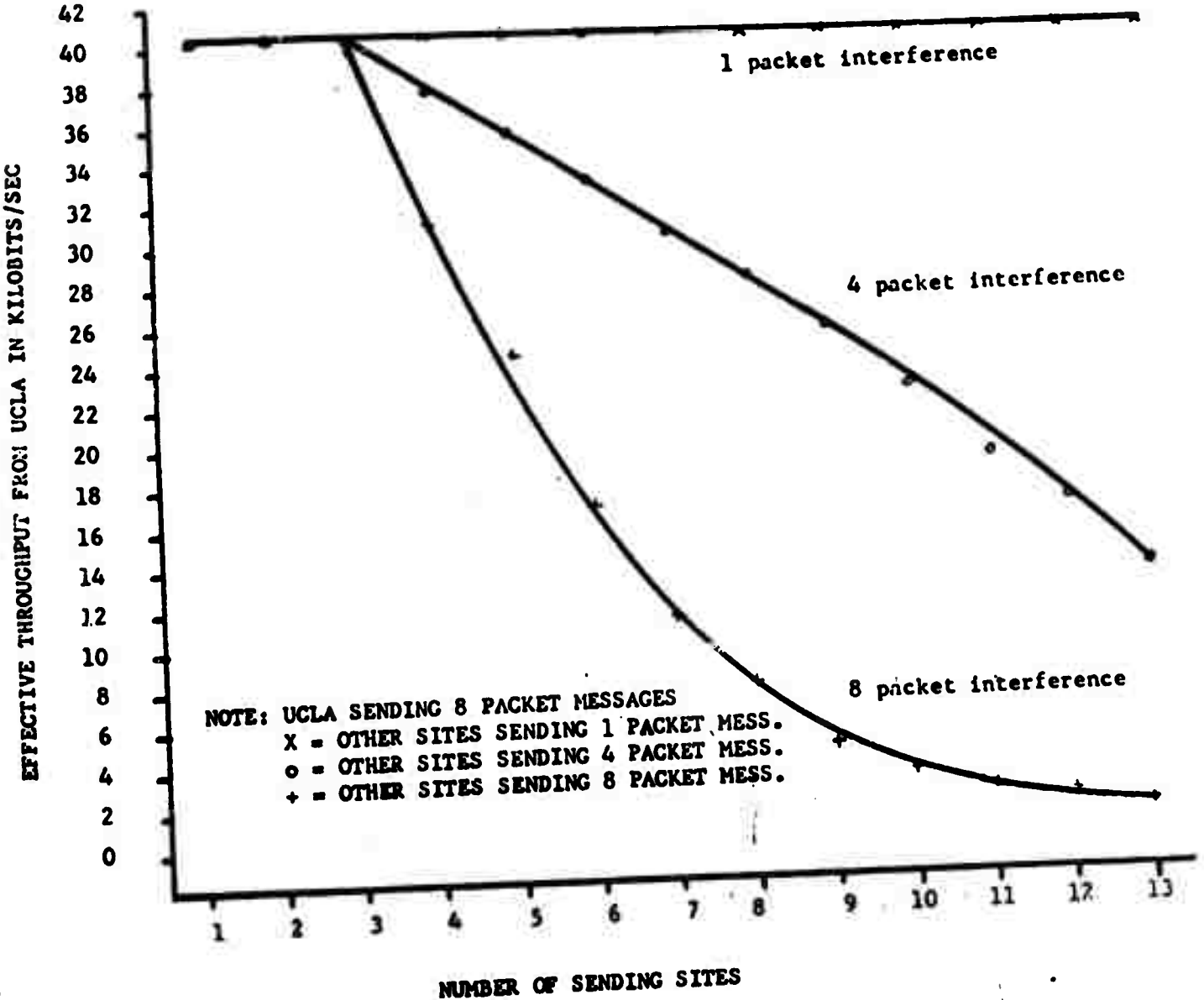


TOTAL THROUGHPUT FROM UCLA VERSUS NUMBER OF SENDING SITES

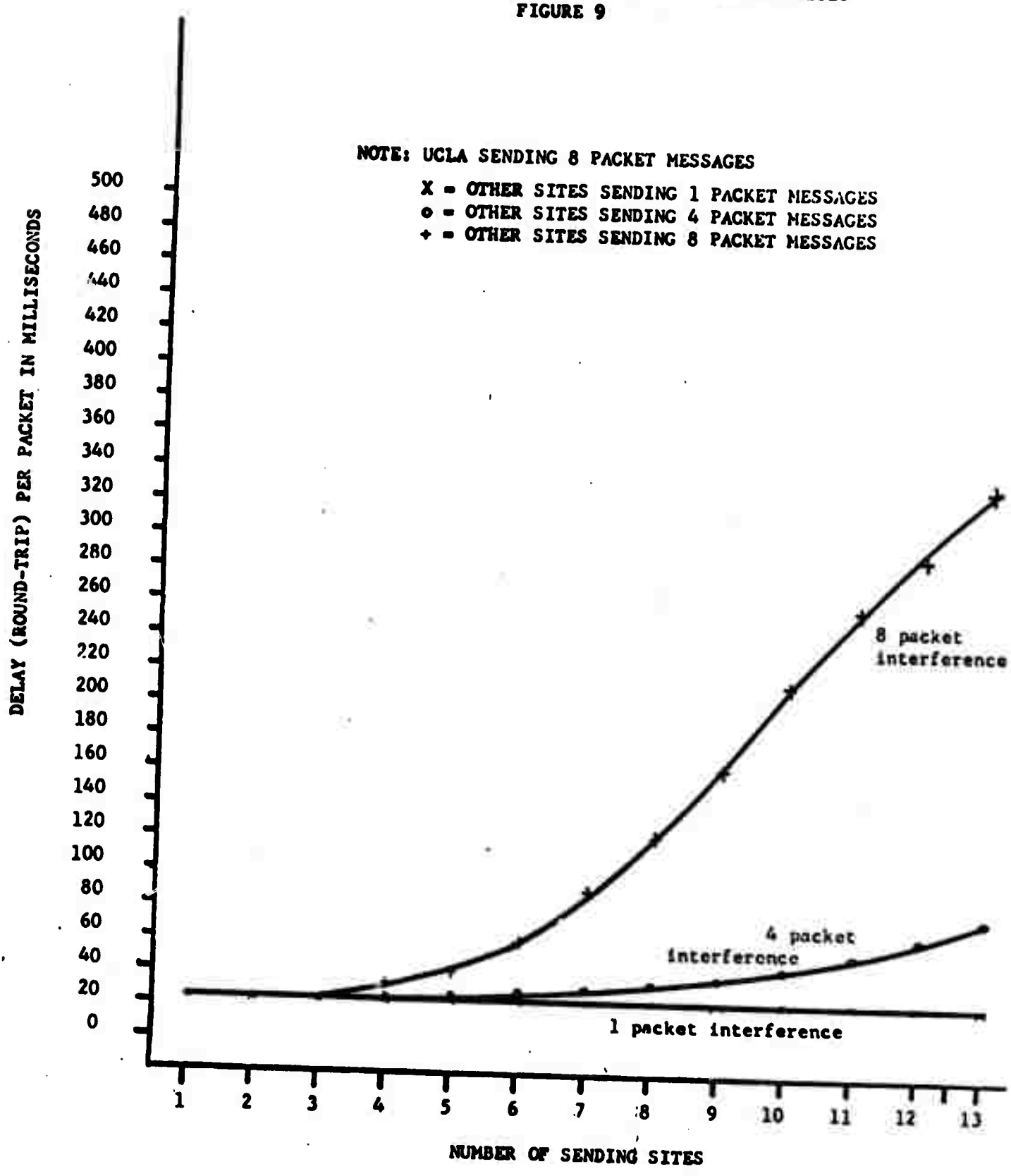
FIGURE 7



EFFECTIVE THROUGHPUT FROM UCLA VERSUS NUMBER OF SENDING SITES
FIGURE 8



DELAY AT UCLA VERSUS NUMBER OF SENDING SITES
FIGURE 9



REFERENCES

BBN1822 "INTERFACE MESSAGE PROCESSOR: Specifications for the Interconnection of a Host and an IMP" Report No. 1822, Bolt Beranek and Newman Inc., 50 Moulton St., Boston, Massachusetts, revised April 1972.

HEART1970 Heart, F.E., R.E. Kahn, S.M. Ornstein, W.R. Crowther, and D.C. Walden, "The Interface Message Processor for the ARPA Computer Network," AFIPS Proceedings of the SJCC, May 1970.

APPENDIX G

FUNCTION-ORIENTED PROTOCOLS FOR THE ARPA COMPUTER NETWORK

by S. Crocker, J. Heafner, R. Metcalfe, and J. Postel

Function-oriented protocols for the ARPA Computer Network

by STEPHEN D. CROCKER
Advanced Research Projects Agency
Arlington, Virginia

JOHN F. HEAFNER
The RAND Corporation
Santa Monica, California

ROBERT M. METCALFE
Massachusetts Institute of Technology
Cambridge, Massachusetts

and

JONATHAN B. POSTEL
University of California
Los Angeles, California

INTRODUCTION

Much has been said about the mechanics of the ARPA Computer Network (ARPANET) and especially about the organization of its communications subnet.[1,2,3,4,5] Until recently the main effort has gone into the implementation of an ARPANET user-level communications interface. Operating just above the communications subnet in ARPANET HOST Computers, this ARPANET interface is intended to serve as a foundation for the organization of function-oriented communications.[6,7] See Figures 1 and 2 for our view of a computer system and the scheme for user-level process-to-process communications. It is now appropriate to review the development of protocols which have been constructed to promote particular substantive uses of the ARPANET, namely function-oriented protocols.

We should begin this brief examination by stating what we mean by the word "protocol" and how protocols fit in the plan for useful work on the ARPANET. When we have two processes facing each other across some communication link, the protocol is the set of their agreements on the format and relative timing of messages to be exchanged. When we speak of a protocol, there is usually an important goal to be fulfilled. Although any set of agreements between cooperating (i.e., communicating) processes is a protocol, the protocols of interest are those which are constructed for general application by a large population of processes in solving a large class of problems.

In the understanding and generation of protocols there are two kinds of distinctions made. Protocols in the ARPANET are *layered* and we speak of high or low level protocols. High level protocols are those most closely matched to functions and low level protocols deal with communications mechanics. The lowest level software protocols in the ARPANET involve reliable message exchange between ARPANET Interface Message Processors (IMPs).[2,5] A high level protocol is one with primitives closely related to a substantive use. At the lowest levels the contents of messages are unspecified. At higher levels, more and more is stated about the meaning of message contents and timing. The layers of protocol are shown in Figure 3.

A second way of structuring sets of protocols and their design is bound up in the word *factoring*. At any level of protocol are sets of format and timing rules associated with particular groupings of agreements. In the IMPs we find certain protocols pertaining to error handling, while others to flow control, and still others to message routing. At the ARPANET's user-level communications interface there are, among others, separable protocols associated with establishing connections and logical data blocking. These protocols do not nest, but join as modules at the same level.

Before moving on to consider the higher level function-oriented protocols, let us first make a few statements about underlying protocols. There are three lower level software protocols which nest in support of the user-level communications interface for the ARPANET. The lowest of these is the IMP-IMP protocol which provides for reliable communication among IMPs. This protocol handles transmission-error detection and correction, flow control to avoid message congestion, and routing. At the next higher level is the IMP-HOST protocol which provides for the passage of messages between HOSTs and IMPs in such a way as to create virtual communication paths between HOSTs. With IMP-HOST protocol, a HOST has operating rules which permit it to send messages to specified HOSTs on the ARPANET and to be informed of the dispensation of those messages. In particular, the IMP-HOST protocol constrains HOSTs in their transmissions so that they can make good use of available

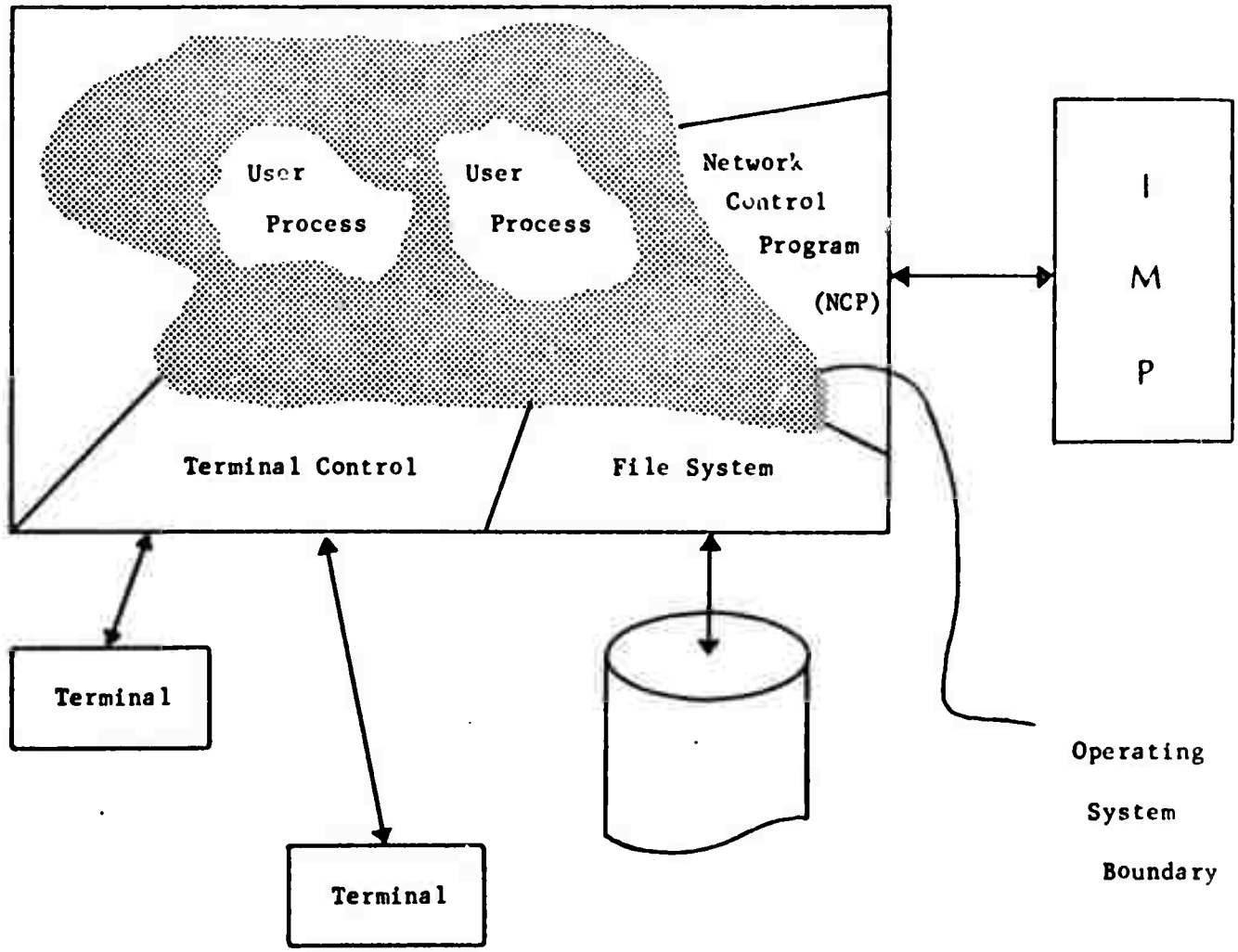


Figure 1—Our view of a computer system

communications capacity without denying such availability to other HOSTs.

The HOST-HOST protocol, finally, is the set of rules whereby HOSTs construct and maintain communication between processes (user jobs) running on remote computers. One process requiring communications with another on some remote computer system makes requests on its local supervisor to act on its behalf in establishing and maintaining those communications under HOST-HOST protocol.

In constructing these low levels of protocol it was the intention to provide user processes with a general set of useful communication primitives to isolate them from many of the details of operating systems and communications. At this user-level interface function-oriented protocols join as an open-ended collection of modules to make use of ARPANET capabilities.

The communications environment facing the designers of function-oriented protocols in the ARPANET is essentially that of a system of one-way byte-oriented connections. Technically speaking, a "connection" is a pair: a "send socket" at one end and a "receive socket" at the other. Primitives provided at the user-level interface include:

1. Initiate connection (local socket, foreign socket),
2. Wait for connection (local socket),
3. Send, Receive (local socket, data),
4. Close (local socket),
5. Send interrupt signal (local socket).

Processes in this virtual process network can create connections and transmit bytes. Connections are subject to HOST-HOST flow control and the vagaries of timing in a widely distributed computing environment, but care has been taken to give user processes control over their communications so as to make full use of network parallelism and redundancy. The kind of agreements which must be made in the creation of

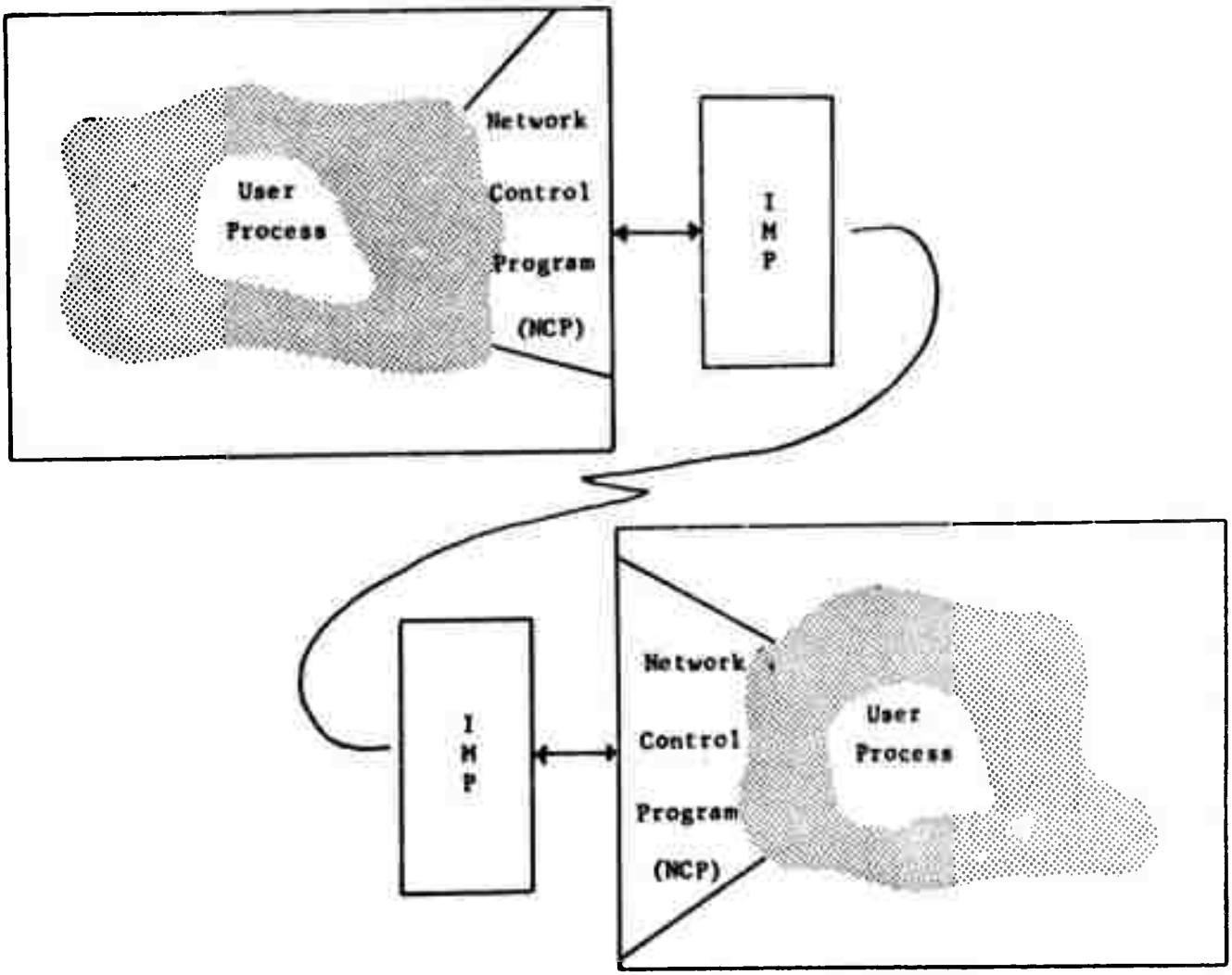


Figure 2—Two communicating processes

function-oriented protocols relate to rules for establishing connections, to the timing rules which govern transmission sequences, and to the content of the byte-streams themselves.

USE OF REMOTE INTERACTIVE SYSTEMS

The application which currently dominates ARPANET activity is the remote use of interactive systems. A Telecommunications Network (TELNET) protocol is followed by processes cooperating to support this application.[8] A user at a terminal, connected to his local HOST, controls a process in a remote HOST as if he were a local user of the remote HOST. His local HOST copies characters between his terminal and TELNET connections over the ARPANET. We refer to the HOST where the user sits as the *using HOST*, and to the remote HOST as the *serving HOST*. See Figure 4.

At the using HOST, the user must be able to perform the following functions through his TELNET user process ("user-TELNET"):

1. Initiate a pair of connections to a serving HOST,
2. Send characters to the serving HOST,
3. Receive characters from the serving HOST,
4. Send a HOST-HOST interrupt signal,
5. Terminate connections.

The user-TELNET needs to be able to distinguish between (1) commands to be acted on locally and (2) input intended for the serving HOST. An escape character is reserved to mark local commands. Conventions for the ARPANET Terminal IMP (TIP) user-TELNET are typical.[9]

In most using HOSTs, the above functions are provided by a user-TELNET which is a *user-level program*. A minimal user-TELNET need only implement the above functions, but several additional support func-

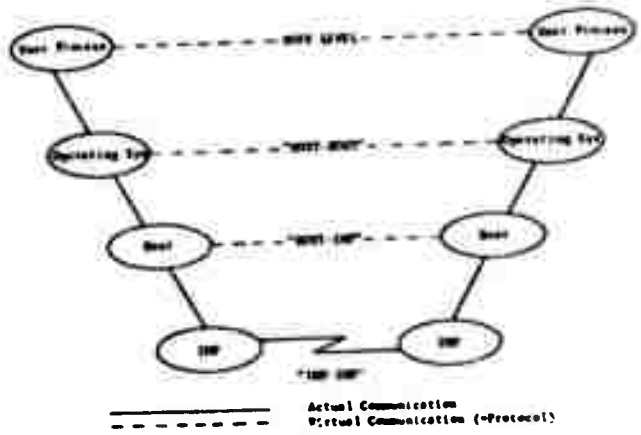


Figure 3—The layers of protocol

tions are often provided (e.g., saving a transcript of a session in a local file, sending a file in place of user-typed input, reporting whether various HOSTs are or have been up).

In the serving HOST it is desirable that a process controlled over the ARPANET behave as it would if controlled locally. The cleanest way to achieve this goal is to generalize the terminal control portion (TCP) of the operating system to accept ARPANET terminal interaction. It is unpleasant to modify any portion of a working computer system and modification could be avoided if it were possible to use a non-supervisor process (e.g., "server-TELNET" or "LOGGER") to perform the job creation, login, terminal input-output, interrupt, and logout functions in exactly the same way

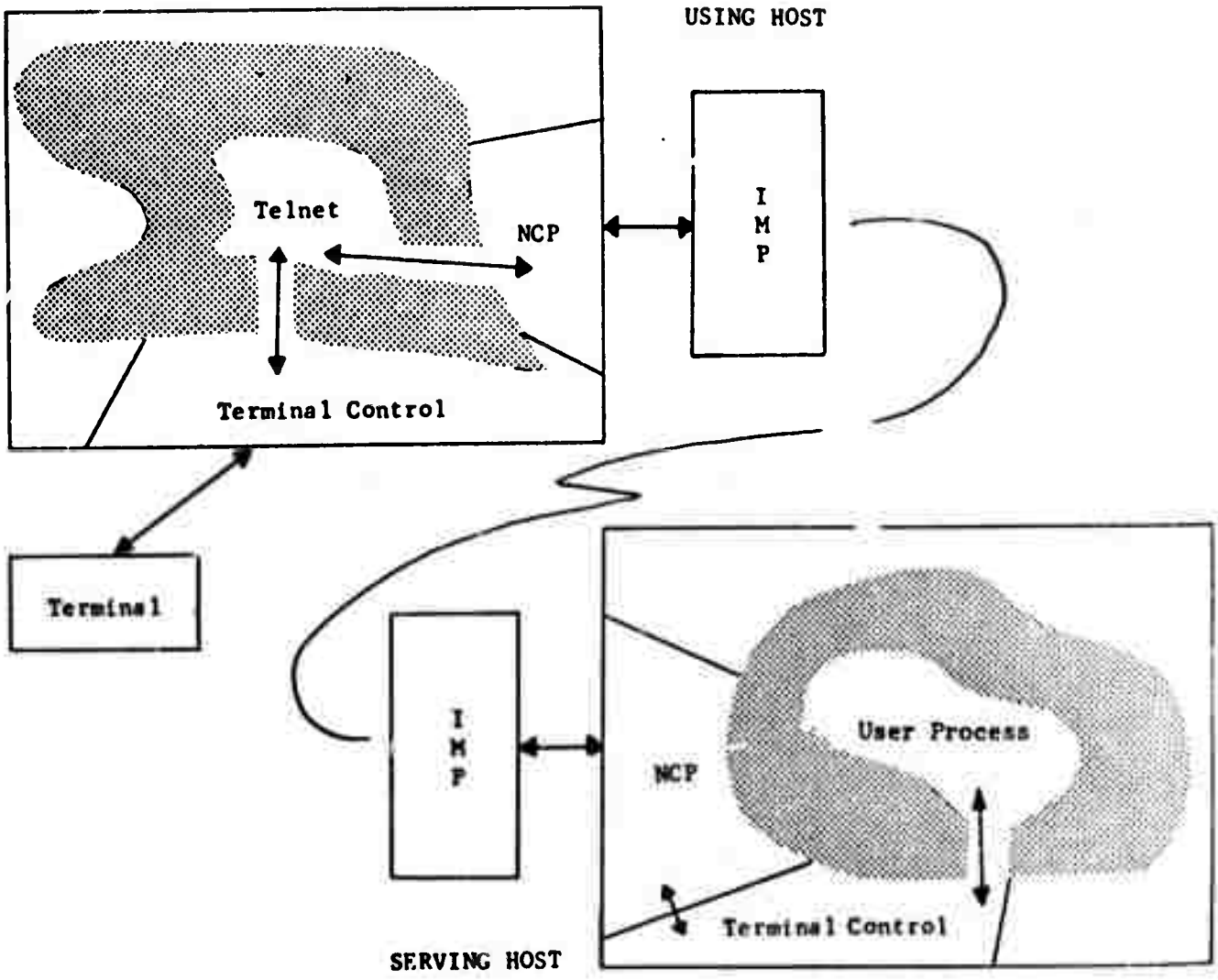


Figure 4—Data flow for remote interactive use

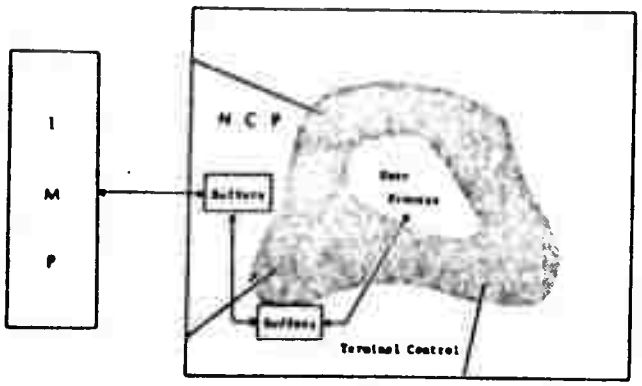


Figure 5—Data flow scheme for server

as a direct console user. Prior to the development of the ARPANET, no operating system provided these functions to non-supervisor processes in anywhere near the required completeness. Some systems have since been modified to support this generalized job control scheme. See Figures 5 and 6.

Efforts to standardize communications in the TELNET protocol focused on four issues: character set, echoing, establishing connections, and attention handling.

The chosen character set is 7-bit ASCII in 8-bit fields with the high-order bit off. Codes with the high-order bit on are reserved for TELNET control functions. Two such TELNET control function codes are the "long-space" which stands for the 200 millisecond space generated by the teletype BREAK button, and the synchronization character (SYNCH) discussed below in conjunction with the purpose of the TELNET interrupt signal.

Much controversy existed regarding echoing. The basic problem is that some systems expect to echo, while some terminals always echo locally. A set of conventions and signals was developed to control which side of a TELNET connection should echo. In practice, those systems which echo have been modified to include provision for locally echoing terminals. This is a non-trivial change affecting many parts of a serving HOST. For example, normally echoing server HOSTs do not echo passwords so as to help maintain their security. Terminals which echo locally defeat this strategy, how-

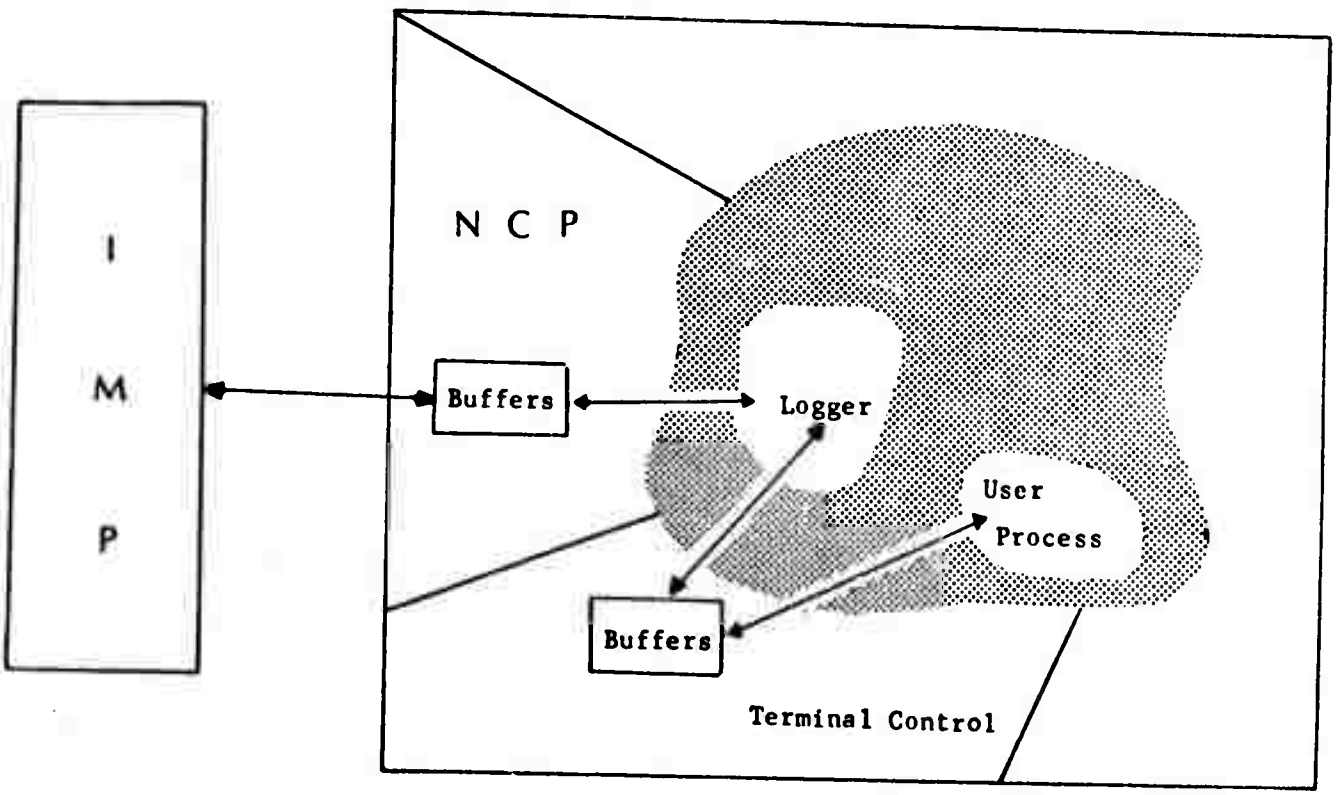


Figure 6—Alternate data flow scheme for a server

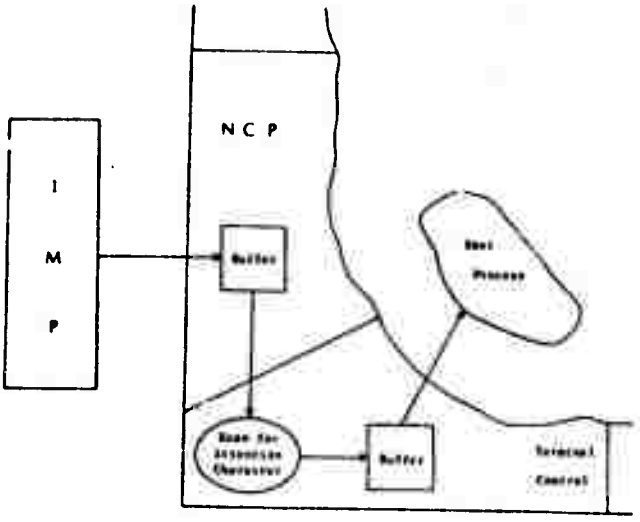


Figure 7—Data flow and processing of the character input stream

ever, and some other protection scheme is necessary. Covering the password with noise characters is the usual solution.

The HOST-HOST protocol provides a large number of sockets for each HOST, but carefully refrains from specifying which ones are to be used for what. To establish communication between a user-TELNET and a server-TELNET some convention is required. The Initial Connection Protocol (ICP)[10] is used:

1. Connection is initiated from a user-TELNET's receive socket to a serving HOST's socket 1 (a send socket).
2. When the initial connection is established, the serving HOST sends a generated socket number and closes the connection. This socket number identifies an adjacent socket pair at the serving HOST through which the user-TELNET can communicate with a server-TELNET.
3. TELNET connections are then initiated between the now specified pairs of sockets. Two connections are used to provide bi-directional communication.

Note that socket 1 at the serving HOST is in use only long enough to send another socket number with which to make the actual service connections.

One of the functions performed by a terminal control program within an operating system is the scanning of an input stream for attention characters intended to stop an errant process and to return control to the executive. Terminal control programs which buffer input sometimes run out of space. When this happens to a local terminal's input stream, a "bell" or a question mark is echoed and the overflow character discarded, *after checking to see if it is the attention character.* See Figure 7. This strategy works well in practice, but it depends rather strongly on the intelligence of the human user, the invariant time delay in the input transmission system, and a lack of buffering between type-in and attention checking. None of these conditions exists for interactive traffic over the net: The serving HOST cannot control the speed (except to slow it down) or the buffering within the using HOST, nor can it even know whether a human user is supplying the input. It is thus necessary that the terminal control program or server-TELNET not, in general, discard characters from a network input stream; instead it must suspend its acceptance of characters via the HOST-HOST flow control mechanism. Since a HOST may only send messages when there is room at the destination, the responsibility for dealing with too much input is thus transferred back to the using HOST. This scheme assures that no characters accepted by the using HOST are inadvertently lost. However, if the process in the serving HOST stops accepting input, the pipeline of buffers between the user-TELNET and remote process will fill up so that attention characters cannot get through to the serving executive. In the TELNET protocol, the solution to this problem calls for the user-TELNET to send, on request, a HOST-HOST interrupt signal forcing the server-TELNET to switch input modes to process network input for attention characters. The server-TELNET is required to scan for attention characters in its network input, even if some input must be discarded while doing so. The effect of the interrupt signal to a server-TELNET from its user is to cause the buffers between them to be emptied for the priority processing of attention characters.

To flip an attention scanning server-TELNET back into its normal mode, a special TELNET synchronization character (SYNCH) is defined. When the server-TELNET encounters this character, it returns to the strategy of accepting terminal input only as buffer space permits. There is a possible race condition if the SYNCH character arrives before the HOST-HOST interrupt signal, but the race is handled by keeping a count of SYNCHs without matching signals. Note that attention characters are HOST specific and may be any of 129 characters—128 ASCII plus "long space"—while SYNCH is a TELNET control character recognized by all server-TELNETs. It would not do to use the HOST-HOST signal alone in place of the signal-SYNCH combination in attention processing, because the position of the SYNCH character in the TELNET input stream is required to determine where attention processing ends and where normal mode input processing begins.

FILE TRANSFER

When viewing the ARPANET as a distributed computer operating system, one initial question is that of how to construct a distributed file system. Although it is constructive to entertain speculation on how the ultimate, automated distributed file system might look, one important first step is to provide for the simplest kinds of explicit file transfer to support early substantive use.

During and immediately after the construction of the ARPANET user-level process interface, several *ad hoc* file transfer mechanisms developed to provide support for initial use. These mechanisms took two forms: (1) use of the TELNET data paths for text file transfer and (2) use of raw byte-stream communication between compatible systems.

By adding two simple features to the user-TELNET, text file transfer became an immediate reality. By adding a "SCRIPT" feature to user-TELNETS whereby all text typed on the user's console can be directed to a file on the user's local file system, a user need only request of a remote HOST that a particular text file be typed on his console to get that file transferred to his local file system. By adding a "SEND-FILE" feature to a user-TELNET whereby the contents of a text file can be substituted for console type-in, a user need only start a remote system's editor as if to enter new text and then send his local file as type-in to get it transferred to the remote file system. Though crude, both of these mechanisms have been used with much success in getting real work done.

Between two identical systems it has been a simple matter to produce programs at two ends of a connection to copy raw bits from one file system to another. This mechanism has also served well in the absence of a more general and powerful file transfer system.

Ways in which these early *ad hoc* file transfer mechanisms are deficient are that (1) they require explicit and often elaborate user intervention and (2) they depend a great deal on the compatibility of the file systems involved. There is an on-going effort to construct a File Transfer Protocol (FTP)[11,12] worthy of wide implementation which will make it possible to exchange structured sequential files among widely differing file systems with a minimum (if any) explicit user intervention. In short, the file transfer protocol being developed provides for the connection of a file transfer user process ("user-FTP") and file transfer server process ("server-FTP") according to the Initial Connection Protocol discussed above. See Figure 8. A user will be able to request that specific file manipulation operations be performed on his behalf. The File Transfer Protocol will support file operations including (1) list remote directory, (2) send local file, (3) retrieve remote file, (4) rename remote file, and (5) delete remote file.

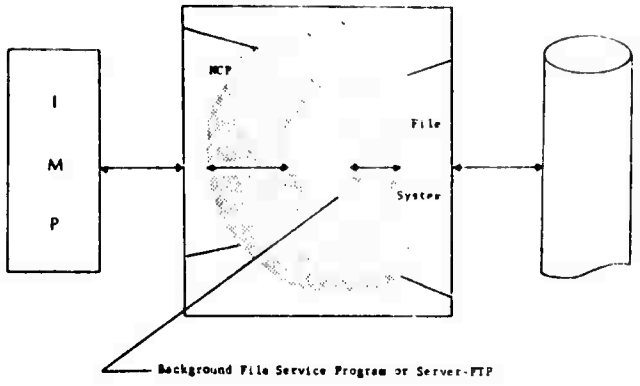


Figure 8—Data flow for file transfer

It is the intention of the protocol designers to regularize the protocol so that file transfer commands can be exchanged by consoles file transfer jobs engaged in such exotic activities as automatic back-up and dynamic file migration. The transfers envisioned will be accompanied with a Data Transfer Protocol (DTP)[11] rich enough to preserve sequential file structure and in a general enough way to permit data to flow between different file systems.

USING THE ARPANET FOR REMOTE JOB ENTRY

A very important use of the ARPANET is to give a wide community of users access to specialized facilities. One type of facility of interest is that of a very powerful number-cruncher. Users in the distributed ARPANET community need to have access to powerful machines for compute-intensive applications and the mode of operation most suited to these uses has been batch Remote Job Entry (RJE). Typically, a user will generate a "deck" for submission to a batch system. See Figure 9. He expects to wait for a period on the order of tens of minutes or hours for that "deck" to be processed, and then to receive the usually voluminous output thereby generated. See Figure 10.

As in the case of file transfer, there are a few useful *ad hoc* ARPANET RJE protocols. A standard RJE protocol is being developed to provide for job submission to a number of facilities in the ARPANET. This protocol is being constructed using the TELNET and File Transfer protocols. A scenario which sketches how the protocol provides the RJE in the simplest, most explicit way is as follows:

Via an ARPANET RJE process, a user connects his

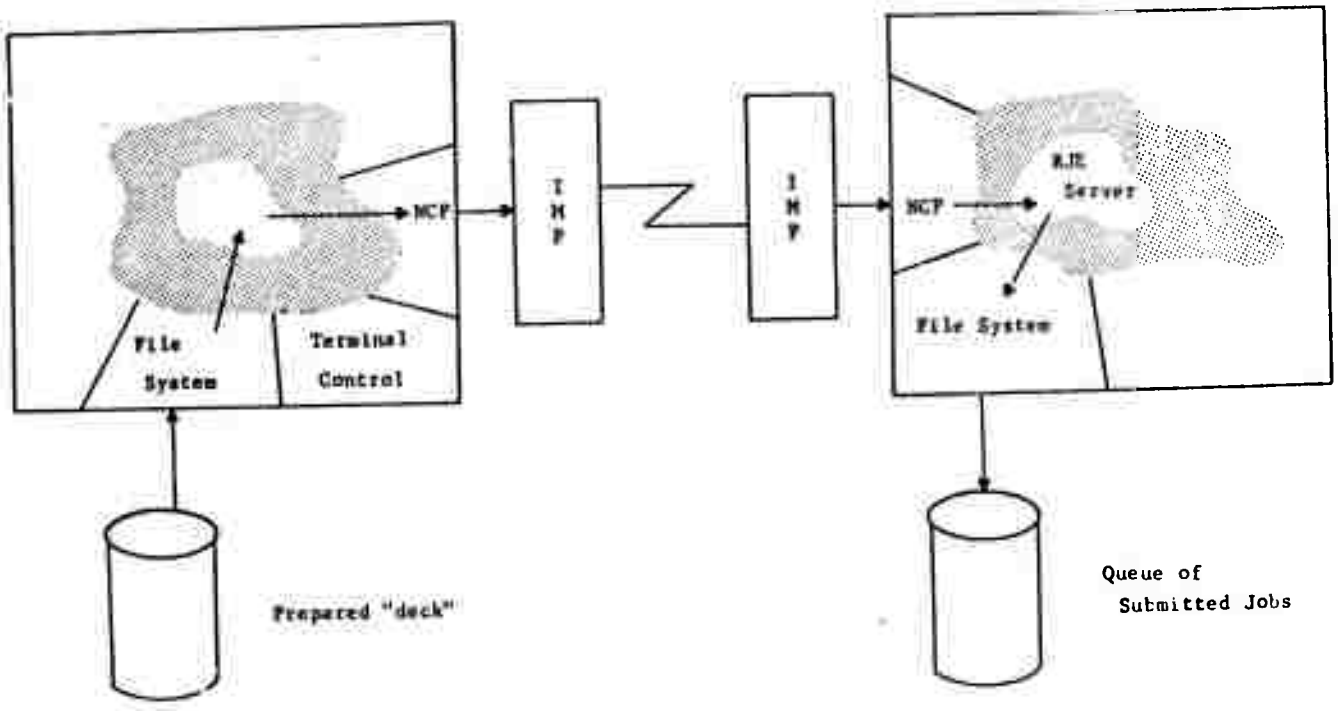


Figure 9—Submission of RJE input

terminal to an RJE server process at the HOST to which he intends to submit his job "deck." Through a short dialogue, he establishes the source of his input and initiates its transfer using the File Transfer Protocol. At some later time, the user reconnects to the appropriate RJE server and makes an inquiry on the

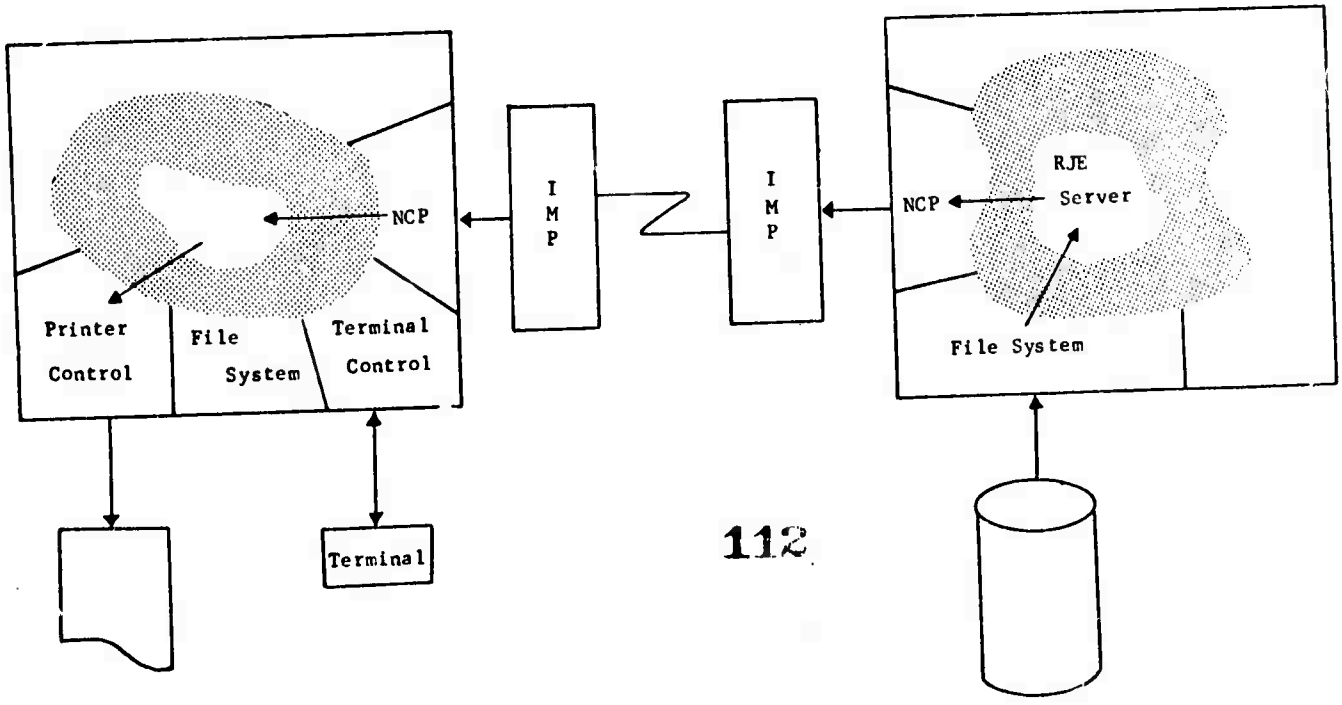


Figure 10—Retrieval of RJE output

status of his job. When notified that his input has been processed, he then issues commands to the serving HOST to transfer his output back.

We can of course imagine more automatic ways of achieving these same functions. A user might need only type a job submission command to his local system. Automatically and invisibly, then, the local system would connect and converse with the specified RJE server causing the desired output to later appear in the users file area or perhaps on a local line printer. The intention is to design the RJE protocol so that the explicit use can start immediately and the more automatic RJE systems can be built as desired.

OTHER PROTOCOLS AND CONCLUSIONS

One of the more difficult problems in utilizing a network of diverse computers and operating systems is that of dealing with incompatible data streams. Computers and their language processors have many ways of representing data. To make use of different computers it is necessary to (1) produce a mediation scheme for each incompatibility or (2) produce a standard representation. There are many strong arguments for a standard representation, but it has been hypothesized that if there were a simple way of expressing a limited set of transformations on data streams, that a large number of incompatibilities could be resolved and a great deal of computer-computer cooperation expedited.

The bulk of protocol work is being done with the invention of standard representations. The TELNET protocol, as discussed, is founded on the notion of a standard terminal called the Network Virtual Terminal (NVT). The File Transfer Protocol is working toward a standard sequential file (a Network Virtual File?). So it is also with less advanced protocol work in graphics and data management.

There is one experiment which is taking the transformational approach to dealing with incompatibilities. The Data Reconfiguration Service (DRS) is to be generally available for mediating between incompatible stream configurations as directed by user-supplied transformations.[13]

ACKNOWLEDGMENTS

Function-oriented protocols have been the principal concern of the ARPANET Network Working Group (NWG). A list of people who have contributed to the development of the protocols discussed would include, Robert Braden, Howard Brodie, Abhay Bhushan, Steve Carr, Vint Cerf, Will Crowther, Eric Harslem, Peggy Karp, Charles Kline, Douglas McKay, Alex McKenzie, John Melvin, Ed Meyer, Jim Michener, Tom O'Sullivan, Mike Padlipsky, Arie Shoshani, Bob Sundberg, Al Vezza, Dave Walden, Jim White, and Steve Wolfe. We would like to acknowledge the contribution of these researchers and others in the ARPA Network Working Group, without assigning any responsibility for the content of this paper.

REFERENCES

1 L G ROBERTS B D WESSLER
Computer network development to achieve resource sharing
AFIPS Conference Proceedings May 1970

2 F E HEART et al
The interface message processor for the ARPA computer network
AFIPS Conference Proceedings May 1970

3 L KLEINROCK
Analytic and simulation methods in computer network design
AFIPS Conference Proceedings May 1970

4 H FRANK I T FRISCH W CHOU
Topological considerations in the design of the ARPA Computer network
AFIPS Conference Proceedings May 1970

5 *Specifications for the interconnection of a Host and an IMP*
Bolt Beranek and Newman Inc Report No 1822
February 1971

6 C S CARR S D CROCKER V G CERF
HOST-HOST communication protocol in the ARPA Network
AFIPS Conference Proceedings May 1970

7 *HOST/HOST protocol for the ARPA Network*
ARPA Network Information Center #7147

8 T O'SULLIVAN et al
TELNET protocol
ARPA Network Working Group Request For Comments (RFC) #158 ARPA Network Information Center (NIC) #6768 May 1971

9 S M ORNSTEIN et al
The Terminal IMP for the ARPA Computer Network
AFIPS Conference Proceedings May 1972

10 J B POSTEL
Official initial connection protocol
ARPA Network Working Group Document #2 NIC #7101 June 1971

11 A K BHUSHAN et al
The data transfer protocol
RFC #264 NIC #7812 November 1971

12 A K BHUSHAN et al
The file transfer protocol
RFC #265 NIC #7813 November 1971

13 R ANDERSON et al
The data reconfiguration service—An experiment in adaptable process/process communication
The ACM/IEEE Second Symposium On Problems In The Optimization Of Data Communications Systems
October 1971